ASSOCIATION RULE MINING OVER MULTIPLE DATABASES:

PARTITIONED AND INCREMENTAL

APPROACHES

The members of the Committee approve the master's
thesis of Hima Valli Kona

Dr. Sharma Chakravarthy
Supervising Professor _____

Dr. Alp Aslandogan _____

Dr. JungHwan Oh _____

ASSOCIATION RULE MINING OVER MULTIPLE DATABASES: PARTITIONED AND INCREMENTAL APPROACHES

by

HIMA VALLI KONA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2003

To My Parents, Family and Friends

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Dr. Sharma Chakravarthy, 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. Alp Aslandogan and Dr. JungHwan Oh for serving on my committee.

I would like to thank Anoop Sanka, Ramanathan Balachandran for maintaining a well-administered research environment and being so helpful at times of need. I am grateful to Pratyush Mishra, Sridhar Reddy, Manu Aery and Laali Elkhalifa for their invaluable help and advice during the implementation of this work. I would like to thank all my friends in the ITLAB. I would also like to thank my friends for their support and encouragement.

I would like to acknowledge the support by the Office of Naval Research, the SPAWAR System Center-San Diego & by the Rome Laboratory grant (AF 26-0201-13), and the NSF (grants IIS-0112914 and IIS-012370) for this research work.

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

November 04, 2003

ABSTRACT

ASSOCIATION RULE MINING OVER MULTIPLE DATABASES:

PARTITIONED AND INCREMENTAL

APPROACHES

Publication No.____

Hima Valli Kona, M.S.

The University of Texas at Arlington, 2003

Supervising Professor: Sharma Chakravarthy

Database mining is the process of extracting interesting and previously unknown patterns and correlations from data stored in Data Base Management Systems (DBMSs). Association rule mining is the process of discovering items, which tend to occur together in transactions. If the data to be mined were stored as relations in multiple databases, instead of moving data from one database to another, a partitioned approach would be appropriate. Also, incremental addition of data to the data set should not necessitate recomputation of rules for the entire data set.

This thesis focuses on partitioned and incremental approaches to association rule mining for data stored in Relational DBMSs. This thesis proposes a partitioning approach

that is very effective for partitioned databases as compared to the main memory partitioned approach. Our approach uses SQL-based K-way join algorithm and its optimizations. A second alternative that trades accuracy for performance is also presented. Our results indicate that, beyond a certain size of data sets, the accuracy is preserved with this approach and results in better performance. The incremental association rule-mining algorithm reduces the task of recomputing the rules each time new data is added to the database. This thesis implements the incremental algorithm using the negative border concept with a number of optimizations. Extensive experiments are performed and results are presented for both partitioned and incremental approaches using IBM DB2/UDB and Oracle 8i.

TABLE OF CONTENTS

ACKNOWLEDGMENTS ..……..…iv

ABSTRACT…….…...v

LIST OF FIGURES ....…….x

LIST OF TABLES ……..…xii

Chapter

1. INTRODUCTION ……...……...1

1.1 Data Mining Techniques.....3

1.1.1 Association Rule.....3

1.1.2 Classification.....3

1.1.3 Clustering 4

1.1.4 Prediction 4

1.1.5 Deviation analysis 4

1.2 Partitioned Approach.....5

1.3 Incremental Mining 5

1.4 Architecture Alternatives.....6

1.5 Background 8

1.6 Focus Of This Thesis 10

2. RELATED WORK...……..…..13

2.1 Association Rule Mining Algorithms 13

2.1.1 Apriori Algorithm 14

2.1.2 Partition Algorithm 16

2.1.3 Parallel Mining of Association Rules 18

2.1.4 Incremental Mining 18

2.2 SQL-OR And SQL-92 Based Approaches 19

2.2.1 Second Pass Optimization 20

2.2.2 Reuse of Item Combinations 21

2.2.3 Vertical-Tid Approach 22

2.3 Multi-Database Mining 23

3. PARTITIONED APPROACH TO ASSOCIATION RULE MINING 27

3.1 Database Approach To Partition Algorithm 27

3.1.1 Methodology for Experiments 28

3.2 Proposed Extensions To Partition Algorithm 31

3.2.1 Approach I 32

3.2.2 Approach II 39

4. INCREMENTAL ASSOCIATION RULE MINING 47

4.1 Incremental Updation Of Frequent Itemsets 47

4.2 Performance Evaluation 56

5. OTHER CONTRIBUTIONS 62

5.1 Configuration File 62

5.2 Writing Log File 65

6. CONCLUSIONS AND FUTURE WORK ……....69

REFERENCES ……...…71

BIOGRAPHICAL INFORMATION…… 74

LIST OF FIGURES

Figure		Page
1.1	Architectural Alternatives	6
2.1	A Multi-Database Environment	24
3.1	Performance Of TIDLIST Approach On T5I2D1000K Dataset	29
3.2	Time Taken For TIDLIST Creation For Different Datasets	30
3.3	Data Transfer Using Approach I	33
3.4	Performance Comparison Of TIDLIST And Approach I	35
3.5	Performance Of TIDLIST And Approach I For T5I2D500K	36
3.6	Data Transfer Using Approach I For 3 Partitions	37
3.7	Performance Comparison Of T5I2D500K For TIDLIST And Approach I	38
3.8	Data Transfer In Approach II	41
3.9	Performance Comparison Of All The Approaches With 2 Partitions	41
3.10	Performance Comparison Of All The Approaches With 3 Partitions	42
3.11	Error Analysis	45
3.12	Comparing performance of Approach I and II	46
4.1	Incremental Mining Algorithm	49
4.2	Updated Database	50
4.3	Frequent Itemsets And Negative Border In DB	51

4.4 Frequent Itemsets In The New Transactions 52
4.5 Case 1 For Incrementally Updating Frequent Itemsets 53
4.6 Case 2 For Incrementally Updating Frequent Itemsets 53
4.7 Case 3 For Incrementally Updating Frequent Itemsets 54
4.8 Performance For T5I2D1000K On Oracle 57
4.9 Performance Of T5I2D100K On Oracle 58
4.10 Performance Of T5I2D500K On Oracle 60

LIST OF TABLES

Table Page

3.1 Notations Used For Partitioned Approach.....31

3.2 Data Transferred Using Approach I.....39

3.3 Data Transfer For Approach II.....43

3.4 Comparison Of Data Transfer For Approach I And Approach II.....43

4.1 Notations Used In Incremental Approach.....48

CHAPTER 1

INTRODUCTION

Database Management Systems have continually evolved from primitive file systems to sophisticated and powerful relational and object oriented models. Present day systems implement various constructs in the form of query optimizing modules, event-condition-action rules to trigger events of interest and other mechanisms that have made their use imperative in most applications. The implicit and unknown patterns in the underlying data can be effectively utilized in decision-making. The process of gleaning important information from data is known as Data Mining. Architectures and techniques for optimizing mining algorithms for relational as well as object oriented databases are being explored with a view to tightly integrate mining into data warehouses. A multi-database system [1, 2] is a federation of autonomous and heterogeneous database systems. Most of the organizations today have multiple data sources distributed at different locations, which need to be analyzed to generate interesting patterns and rules. An effective way to deal with multiple data sources (where data to be mined is distributed among several relations on different database management systems (DBMSs)) is to mine the association rules at different sources and forward the rules to a centralized system rather than sending the data to be mined which is likely to be very large. This would provide a great boost to the database and information

industry, and make available large data repositories for transaction management, information retrieval, and data analysis

A data warehouse is an information repository that is a complete and consistent store of data obtained from various sources. Data mining [1, 3, 8, 9, 10, 18, 20, 25] has attracted a great deal of attention due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information. The knowledge gained can be used for applications ranging from business management, production control, and market analysis, to engineering design and science exploration. Interactive mining has been proposed as a way to bring decision makers into the loop to enhance the utility of mining and to support goal oriented mining.

Data mining is also known as knowledge discovery in databases [3]. It is the automated extraction of implicit, understandable, previously unknown and potentially useful information from large databases. In other words, data mining is the act of drilling through huge volumes of data to discover relationships, or answer queries too generalized for traditional query tools. In general, data mining tasks can be classified into two categories:

Descriptive mining: It is the process of drawing the essential characteristics or general properties of the data in the database. Clustering, Association and Sequential mining are some of the descriptive mining techniques.

Predictive mining: This is the process of inferring patterns form data to make predictions. Classification, Regression and Deviation detection are predictive mining techniques.

1.1 Data Mining Techniques

1.1.1 Association Rule

Association rule mining [3-6] is a data mining technique used to find interesting associations among a large set of data items. A typical application of association rule mining is market-basket analysis. In market-basket analysis, buying habits of customers are analyzed to find associations between the different items that customers place in their "shopping baskets". The discovery of such associations can help retailers develop marketing and placement strategies as well as plan on logistics for inventory management. Items that are frequently purchased together by customers can be identified. An attempt is made to associate a product "A" with another product "B" so as to infer "whenever A is bought, B is also bought", with high confidence (i.e., the number of times B occurs when A occurs).

1.1.2 Classification

Classification [3, 4] is the process of partitioning a given dataset into disjoint classes using a class attribute. For example, in determining a store location, the success of a store is determined by its neighborhood. The company is interested in identifying neighborhoods that would constitute its primary candidates. A model is built based on the values of all attributes to classify each item into a particular class. The goal of classification is to analyze the training set and to develop an accurate description or model for each class using the attributes presented in the data. Many classifications models have been developed such as neural networks, genetic models, and decision trees etc.

1.1.3 Clustering

Clustering [3] is the process of grouping the data into clusters with high intra-cluster similarity and low inter-cluster similarity. A similarity measure needs to be defined and the quality of the cluster, to a large extent, depends on the appropriateness of the similarity measure for the data set or the domain of application. The technique of clustering, for example, can be used to divide the market into distinct groups, so that each group can be targeted with a different strategy. There are several clustering techniques: partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods.

The basic difference between classification and clustering is that classification is a supervised learning method, which assumes predefined class labels, while clustering is an unsupervised learning method that does not assume any knowledge of classes.

1.1.4 Prediction

Prediction techniques are based on some continuous valued attributes. Previous history of the attributes is used to build the model. This technique is commonly used for predicting product sales.

1.1.5 Deviation analysis

This technique compares current data with previously defined normal values to detect anomalies. Deviation analysis tools are useful in security systems, where authorities can be warned about deviation in resource utilization.

1.2 Partitioned Approach

The partition algorithm is a fast and efficient algorithm for mining association rules in large databases. The algorithm differs from the other mining algorithms in terms of the number of passes it makes over the database. The algorithm makes just 2 passes over the input database to generate the rules. The partition algorithm executes in two phases: In the first phase of the algorithm the database is divided into non-overlapping partitions and each of the partitions are mined individually to generate the local frequent itemsets. At the end of the first phase the local frequent itemsets are merged to generate the global candidate itemsets. In the second phase of the algorithm the support is calculated for all the itemsets in the global candidate itemset and the global frequent itemsets are generated as the set of itemsets, which satisfy the support. The database is scanned completely once in each of the phases of the algorithm. The algorithm can be executed in parallel to utilize the capacity of many processors with each processor generating the rules for a particular set of transactions and merging the frequent itemsets obtained from all the processors.

1.3 Incremental Mining

Association rules represent an important class of knowledge that can be discovered from data warehouses [7-9]. As new data is added, previously discovered rules have to be verified and new rules may have to be added to the knowledge base. Changes to the data can also invalidate existing patterns. The task of deriving new rules for data sets that grow incrementally can be done in several ways. Re-executing the algorithms from scratch each time a database is updated can result in excessive computation and I/O. Re-execution is not

an efficient process, since it ignores previously discovered rules and repeats the work that has already been done. Incremental mining is a useful technique for discovering new rules as the data distribution patterns change, obviating the need for recomputation of old rules.

Incremental mining is done by maintaining the negative border [10] along with the frequent itemsets. The negative border is used to decide when to scan the whole database. The frequent itemsets for the increment database is computed. A full scan of the whole database is required when the negative border of the frequent itemsets expands, that is an itemset outside the negative border gets added to the frequent itemsets or its negative border.

1.4 Architecture Alternatives

Various architecture schemes have been proposed for integrating the mining process with relational database systems. These alternatives are depicted in Figure 1.1 [11] and are described below.

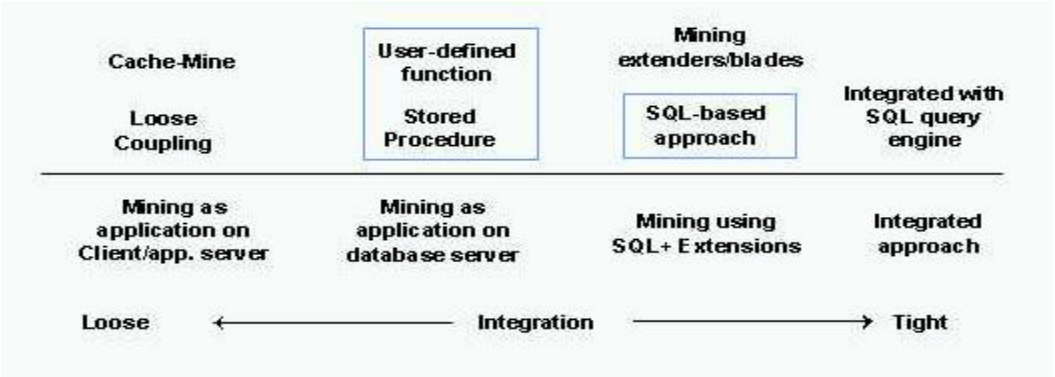


Figure 1.1 Architectural Alternatives

Loose coupling or Cache-based Mining: This is an example of integrating mining applications into the client in a client/server architecture or into the application server in a multi-tier architecture. The mining kernel can be considered as the application server. The data is first fetched from the database and fed to the mining-kernel, which mines and pushes the results back to the database.

In loose coupling, the DBMS runs in a different address space from the mining process. Cache-based mining is a special case of the loose coupling approach, wherein data from the DBMS is read only once and the relevant data is cached into flat files on local disk.

Stored procedures and user-defined functions: This architecture is a representative of the case where the mining logic is embedded as an application on the database server. The applications are executed in the same address space as the DBMS. The flexibility in programming the stored procedure outweighs their development cost.

SQL-based approach: In this approach the mining algorithm is formulated as SQL queries, which are executed by the DBMS query processor. A mining-aware optimizer may be used to optimize these complex, long running queries based on the mining semantics. The DBMS support for check pointing and space management is especially valuable for long running mining algorithms on huge volumes of data.

Integrated Approach: This is the tightest form of integration that has no boundary between simple querying, OLAP, or mining. Mining operators or SQL, extended for mining is optimized by the underlying system without any hints from the user. The long-term goal is to extend the current query optimizers to cover OLAP and mining along with SQL queries.

1.5 Background

The work on association rule mining began with the development of the AIS algorithm [6], and was further modified and extended in [5]. Since then, several attempts have been made to improve the performance of these algorithms. The partition algorithm [12] partitions the data into disjoint groups, processes each individually, and merges the intermediate results. It improves the overall performance by reducing the number of passes needed over the complete database to at most two. The turbo-charging algorithm [13] incorporates the concept of data compression to boost the performance of the mining algorithm. The FP-Tree algorithm [14] builds a special tree structure in main memory to avoid multiple passes over database. However, most of these algorithms are applicable to data stored in flat files. The main characteristic of these algorithms is that they are main memory algorithms. In these algorithms, the data is either read directly from flat files or is first extracted from the DBMS and then processed in main memory. The algorithms implement their own buffer management schemes and the performance varies depending on the specialized data-structures used for buffer management.

A few attempts have been made to build database-based mining approaches. Work in the field of database mining has focused on integrating the mining functions with the database. Various extensions to the SQL have been proposed which overload the SQL with certain mining operators. SETM [15] showed how the data stored in RDBMS can be mined using SQL and the corresponding performance gain achieved by optimizing these queries. The Data Mining Query Language DMQL [16] proposed a collection of such operators for

classification rules, characteristics rule, association rules, discriminant rules, etc. [17] proposed the MineRule operator for generating general/clustered/ordered association rules. [18] presents a methodology for tightly coupled integration of data mining applications with a relational database system. [19] has tried to highlight the implications of various architectural alternatives for coupling data mining with relational database systems. They have also compared the performance of the SQL-based approaches with SQL-OR-based approaches and the case when mining is done outside the database address space. The Incremental Mining algorithm [7-9] is another useful technique for speeding up the mining process when new data is added. [20] has developed an association rule visualization system, which includes a tabular form and a three-dimensional graphics to display the rules. [21] has formulated SQL queries to implement association rule mining algorithms and also compared and contrasted the performance of SQL-92 and SQL-OR approaches based on their performance over synthetically generated datasets. Some of the earlier research has focused on the development of SQL-based formulations for association rule mining. Most of these algorithms use the apriori algorithm directly or indirectly with certain modifications of the same. [19] and [8] deal with the SQL implementation of the apriori algorithm and have compared some of the optimizations to the basic k-way join algorithm for association rule mining but the relative performances and possible combinations for optimizations were not explored. [22] deals with the mapping of the arbitrary relations into the (tid,item) format and remapping them back to the original values. [21, 23, 24] deals with the performance evaluation of the SQL-92 and SQL-OR approaches. [1] Deals with a multi-database mining strategy to develop local pattern analysis for identifying novel and useful patterns. [2]

presents a weighting model for synthesizing high-frequency association rules from different data sources.

To make the implementation operating system independent, we have used Java and JDBC API's [25, 26]. For the purpose of evaluation, the experiments have been performed on Oracle 8i and IBM DB2/UDB.

1.6 Focus Of This Thesis

With increase in the use of RDBMS to store and manipulate data, mining directly on RDBMSs would be an advantage. Main memory imposes a limitation on the size of dataset that can be processed. In addition, data stored in a database has to be siphoned out into a flat file for processing. However, if mining is done directly over an RDBMS, the user/application can be freed from data size considerations, as this would be taken care of by the underlying buffer management system. In database mining, we assume that the data is already stored in tables in an underlying DBMS and use the SQL provided by the RDBMS for mining to produce association rules. Building mining algorithms to work on RDBMSs also provides the advantage of mining over very large datasets as RDBMSs have been built to manage such large volumes of data. File based mining algorithms are those that work on data outside the database. They generally have an upper limit on the number of transaction that can be mined. For example, the DBMiner has an upper limit of 64K on the number of unique transactions that it can process for mining. With the users having a choice of RDBMS to use for their applications, the mining algorithms should be developed using such accepted standards so that the underlying system is not a limitation and should be portable to other RDBMSs.

Keeping this in mind, focus has been placed on the use of SQL. The arbitrary relations are mapped [22] to the (Tid, item) format and reconversion is done at the end of the rule generation. Data may be stored in multiple databases and the data in each of the databases may get updated frequently and independently. In this case either the intermediate results or the input data has to be transferred to a single database to perform mining.

The goal of this thesis is to study approaches for mining association rules over multiple databases. In an organization, data is generally distributed over multiple databases (typically 2 or 3). One of the solutions to mining data in multiple databases is to move the relations to a single database and do mining. The other solution would be to apply a partitioned or an incremental approach to perform mining. In this case, some intermediate data has to be transferred to one database (most likely one o the databases) to combine the results obtained independently at different sources. This thesis mainly focuses on two approaches for association rule mining over data stored in multiple relations in one or more databases. First is the partition approach [12], which mines the data in partitions and merges the results finally. The second is the incremental mining approach [8] that helps to update and manage the rules each time new data is added or existing data is deleted from the warehouse. These approaches may be extended to suit a multi-database environment that has autonomous and heterogeneous data sources. Due to the lack of availability of real datasets, synthetic datasets (generated by the program developed at IBM Almaden) have been used for performance evaluation. Nevertheless, the results are useful (as they are only based on cardinality, support and underlying RDBMS, not on the semantics of the data set) in understanding the approaches.

The rest of this thesis is organized as follows. CHAPTER 2 introduces the association rule mining algorithms and their SQL formulations. CHAPTER 3 discusses the partition-based approach for association rule mining. It covers in detail the implementation of the algorithm using the k-way join approach for support counting. The optimizations proposed and their performance analyses are presented. CHAPTER 4 presents incremental mining of association rules. CHAPTER 5 includes extensions done in building this mining tool. CHAPTER 6 concludes the thesis with emphasis on the future work.

CHAPTER 2

RELATED WORK

The outline of this chapter is as follows: We revisit association rule mining briefly in 2.1 and discuss some of the main-memory based mining algorithms such as Apriori algorithm, the partition algorithm and the incremental mining algorithm. 2.2 enumerates the SQL-OR and SQL-92 based approaches for generating the frequent itemsets. 2.3 discusses multi-database mining for analyzing data from different sources.

2.1 Association Rule Mining Algorithms

Association rule mining makes correlation among items that are grouped into transactions, deducing rules that define relationships between item sets. The rules have a user-stipulated support, confidence, and length. Association rule mining has attracted tremendous attention from data mining researchers and as a result several algorithms have been proposed for it [8, 9, 14, 20, 21]. Let $I = \{i_1, i_2, \ldots, i_m\}$ be the collection of all the items and D be the set of database transactions where each transaction T is a set of items such that $T \subseteq I$. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$. An

association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, and $A \cap B = \phi$. They are two terms associated with association rules. These are: *Support* and *Confidence*.

If the support of itemset {AB} is 30%, it means "30% of all the transactions contain both the itemsets – itemset A and itemset B".

$$\text{Support of itemset } \{AB\} = \frac{\text{Count Of the transactions containing the itemsets A and B}}{\text{Total Number of Transactions}}$$

If the confidence of the rule $A \Rightarrow B$ is 70%, it means "70% of all the transactions that contain itemset A also contain itemset B".

$$\text{Confidence of the rule } A \Rightarrow B = \frac{Support(\{AB\})}{Support(\{A\})}$$

An association rule-mining problem is broken down into two steps: 1) Generate all the item combinations (itemsets) whose support is greater than the user specified minimum support. Such sets are called the frequent itemsets and 2) use the identified frequent itemsets to generate the rules that satisfy a user specified confidence. The frequent itemsets generation requires more effort and the rule generation is straightforward.

2.1.1 Apriori Algorithm

The apriori algorithm [6] is based on the above-mentioned steps of frequent itemsets and rule generation phases. Frequent itemsets are generated in two steps. In the first step all possible combination of items, called the candidate itemset (C_k) is generated. In the second step, support of each candidate itemset is counted and those itemsets that have support values greater than the user-specified minimum support form the frequent itemset (F_k). In this

algorithm the database is scanned multiple times and the number of scans cannot be determined in advance. The apriori algorithm is depicted below.

```
F1 = {frequent 1-itemsets}
for (k = 2; Fk-1 ≠ 0; k++) do
     Ck = generate(Fk-1)
     for all transactions t ∈ D do
          Ct = subset(Ck, t)
          for all candidates, c ∈ Ct do
               c.count++
          end for
     end for
     Fk = { c ∈ Ck | c.count ≥ minsup}
end for
Answer = ∪k{Fk}
```

The AprioriTid algorithm [5] uses the above algorithm to determine the candidate itemsets before each pass begins. The interesting feature of this algorithm is that it does not use the database for support counting after the first pass. It uses a set C_k of the form {TID, X_k} where X_k is the potentially large k-itemsets present in the transaction with the identifier TID. For k=1, C_1 will be the database. If a transaction does not contain a k-itemset then C_k will not have an entry for that transaction. Set Oriented Mining, SETM [15] uses the SQL join operation for candidate generation. The candidate itemset with the TID of the generating transaction is stored as a sequential structure, which is used for support counting. The problem with the above two algorithms is that they generate too many candidates that turn out to be small (or not frequent) resulting in wasted effort. AprioriTid is better for the later passes where the size of Ck is small when compared to the size of the database.

Based on the above observations the Apriori Hybrid [5] algorithm was proposed. It uses Apriori for the earlier passes and switches to the AprioriTid when the size of C_k becomes small enough to fit in memory.

2.1.2 Partition Algorithm

The Partition algorithm [12] differs from the Apriori algorithm in terms of the number of database scans. The partition algorithm scans the database at most twice. The algorithm is inherently parallel in nature and can be parallelized with minimal communication and synchronization between the processing nodes The algorithm is divided into 2 phases: i) during the first phase, the database is divided into n non-overlapping partitions and the frequent itemsets for each partition are generated. ii) In the second phase, all the local large itemsets are merged to form the global candidate itemsets and a second scan of the database is made to generate the final counts. The algorithm is depicted below:

```
P = partition_database(D)
n = Number of partitions
// Phase I
for i = 1 to n do begin
     read-in_partition(p_i ∈ P)
     L^i - gen_large_itemsets(p_i)
end
// Merge Phase
for(i = 2; L^j_i ≠ ϕ, j = 1,2....,n; i++) do begin
     C_i^G = ∪_{j = 1,2,...n} L_i^j
end
// Phase II
for i = 1 to n do begin
     read-in_partition(p_i ∈ P)
```

```
    for all candidates c ∈ C^G gen_count(c, p_i)
end
L^G = {c ∈ C^G| c.count ≥ minsup}
Answer = L^G
```

The partition algorithm is designed to generate rules in parallel and utilize the power of a number of processors. It is used to aggregate the power and memory of many processors. The data in the form of a single file is distributed among different processors with each processor generating itemsets for that part of the data in parallel. This would require the passing of intermediate information among processors to generate the global rules. The parallelized partition algorithm, although developed for multiple processors, can be adapted for multiple database scenario where data is distributed over multiple databases.

2.1.3 Parallel Mining of Association Rules

[27] discusses the problem of mining association rules in a shared-nothing multiprocessor. Three algorithms were proposed to explore the spectrum of trade-offs between computation, communication and memory usage as follows:

Count Distribution Algorithm: The count distribution algorithm minimizes the communication at the expense of carrying out redundant duplicate computations in parallel. These communications are carried out on the idle processors. This algorithm does not use the memory of the system effectively.

Data Distribution Algorithm: The data distribution algorithm attempts to utilize the aggregate main memory of the system effectively depending on the number of processors. The downside of this algorithm is that every processor must broadcast its local data to all

other processors in every pass. This algorithm would be viable only on a machine with very fast communication.

Candidate Distribution Algorithm: This algorithm exploits the semantics of the particular problem at hand to reduce the synchronization between the processors and to segment the database based on the patterns the different transactions support.

The count distribution algorithm performed the best among the three algorithms. It exhibited linear scale-up and excellent speed-up and sizeup behavior.

2.1.4 Incremental Mining

The Incremental mining algorithm [7, 8] is used to find new frequent itemsets with minimal recomputation when new transactions are added to or deleted from the transaction database. The algorithm uses the negative border concept for this. The negative border [Toivonen, 1996 #28] consists of all itemsets that were candidates, which did not have the minimum support. During each pass of the apriori algorithm, the set of candidate itemsets C_k is computed from the frequent itemsets F_{k-1} in the join and prune steps of the algorithm. The negative border is the set of all those itemsets that were candidates in the k^{th} pass but did not satisfy the user specified support, that is $(NBd(F_k)) = C_k - F_k$. The algorithm uses a full scan of the whole database only if the negative border of the frequent itemsets expands. The algorithm for updating the frequent itemsets is as follows:

```
function Update-Frequent-Itemset(F^DB, NBd(F^DB),db)

//DB and db denote the number of transactions in the original
database and the increment database respectively.

Compute F^db
for each itemset s ∈ F^DB ∪ NBd(F^DB) do
     t_db(s) = number of transactions in db containing s
     F^DB+ = ϕ
for each itemset s ∈ F^DB do
     if (t^DB(s) + t^db(s)) ≥ minsup * (DB + db)
          then F^DB+ = F^DB+ ∪ s
for each itemset s ∈ F^db do
     if s ∉ F^DB and s ∈ NBd(F^DB) and (t^DB(s) + t^db (s)) ≥
     minsup * (DB + db)
     then F^DB+ = F^DB+ ∪ s

if F^DB ≠ F^DB+ then
     NBd(F^DB+) = negativeborder-gen(F^DB+)
else NBd(F^DB+) = NBd(F^DB)
if F^DB ∪ NBd(F^DB) ≠ F^DB+ ∪ NBd(F^DB+) then
     S = F^DB+
repeat
     compute S = S ∪ NBd(S)
until S does not grow
F^DB+ = {x ∈ S | support(x) ≥ minsup}
//support(x) is the support count of x in DB ∪ db
NBd(F^DB+) = negativeborder-gen(F^DB+)
```

2.2 SQL-OR And SQL-92 Based Approaches

The k-way join approach [8, 21-23] is the SQL-92 approach for support counting. Here in any pass k, k copies of the input table are joined with the candidate itemsets C_k followed by a group by on the itemsets. The k copies of the input table are needed to compare the k items in the candidate itemset C_k with one item from each of the k-copies of the input table. The group by clause on the k items is done to identify all itemsets whose

count is greater than the user specified support value, as frequent items, which are then used for the rule generation phase. The SQL statement used for support counting in the k-way join approach is shown below.

```
Insert    into Fk
Select    item1, … , itemk, count(*)
From      Ck, T t1, … , T tk
Where     t1.item = Ck.item1 and
                  :
          tk.item = Ck.itemk and
          t1.tid = t2.tid and
                  :
          tk-1.tid = tk.tid
Group by item1,  item2, … ,itemk
Having    count(*) > minsup
```

The following are the optimizations that turned out to be the best [21, 23] for the k-way join approach:

2.2.1 Second Pass Optimization

In general, because of the immense size of C_2, the cost of support counting for C_2 is very high. In addition, for candidate sets of length 2, as all the subsets of length 1 are known to be frequent, there is no gain from pruning during candidate generation. Also there are no rules associated with F_1. Hence the process of generating F_1 and then C_2 followed by the support counting phase can be replaced by directly generating F_2. F_2 is generated by joining two copies of the input table such that, the item from first copy of the input table is less than

the item from the second copy of the input table and both items belong to same transaction. The SQL for the same is as follows:

```
Insert    into F2 select t1.item, t2.item, count(*)
From      InputTable T1, InputTable T2
Where     T1.tid = T2.tid and T1.item < T2.item
Group by  T1.item, T2.item.
Having    count(*) > minsup
```

2.2.2 Reuse of Item Combinations

This optimization aims to reduce the cost of support counting, in any pass k, by avoiding the join of k copies of input table with the set of candidate itemsets C_k. Joining k copies of the input table is avoided by materializing the frequent itemsets obtained from a particular transaction in pass k-1, and using it for support counting in the k^{th} pass. This approach proves to be very effective for cases where the length of the frequent itemset is large since the sequence of joins done in the earlier passes are avoided. So in k^{th} pass for support counting, a relation $Comb_{k,}$ having the following attributes (tid, $item_1$, $item_2$, …, $item_k$) is created. The tuples in $Comb_k$ is the result of the join between $Comb_{k-1}$, T and C_k to select all those transactions in T which contains 1-extensions to the frequent itemsets of length k-1. The SQL for this is given below:

```
Insert    into Comb_k
Select    T1.tid, T1.item_1, T1.item_2,…, T1.item_k-1,T2.item
From      C_k, Comb_k-1 T1, T T2
```

```
Where     T1.item1 = Ck.item1 and
                  :
                  :
          T1.itemk-1 = Ck.itemk-1 and
          T2.item = Ck.itemk and
          T1.tid = T2.tid
```

F_k is then generated from $Comb_k$ by grouping on k items ($item_1$, $item_2$, …, $item_k$) and selecting those that satisfy the minimum support criteria. The SQL for this given below:

```
Insert    into Fk
Select    item1, item2, …, itemk
From      Combk
Group by  item1, item2, …, itemk
Having    count(*) > minsup
```

2.2.3 Vertical-Tid Approach

The Vertical-Tid approach [24] uses SQL-OR constructs (such as CLOBs) for better representation of input data. For Oracle, all stored procedures have been implemented as a Java stored procedures and for IBM DB2/UDB, the same has been implemented as user defined functions (or UDFs) using Java.

Here, the representation of input data is changed and the transactions are inserted in a different relation (TidListTable) having the following attributes: (Item, TidList). For every unique item id in the input dataset, the TidListTable has only one tuple. This tuple represents the item id and the list of all the transactions in which that item was bought. Each list of transactions is represented as a CLOB and stored in the TidList column of the TidListTable.

For the purpose of support counting, procedures are used to read these CLOBs and for each item combination (itemset), the numbers of same transaction ids that are present in the TidList of each item id in that itemset are counted. The SQL for generation of frequent itemsets is given below.

```
Insert    into Fk
Select    item1, item2, …, itemk
From      (Select   item1, item2,…, itemk,
                    CountAndK(I1.TidList,I2.TidList, … ,
                    Ik.TidList) as cnt
          From      Ck, TidListTable I1, TidListTable I2,…,
                    TidListTable Ik,
          Where     Ck.item1 = I1.item And
                    Ck.item2 = I2.item And
                              :
                              :
                    Ck.itemk = Ik.item) as temp
Where     cnt > minsup.
```

Here **CountAndK** is a procedure that in pass k, accepts k TidLists and returns the count of transactions that are common to each of them.

2.3 Multi-Database Mining

Many organizations end up using multiple databases due to acquisitions and merger. These are used in a federated manner and are independently, maintained. If one were to mine on data present in multiple databases, there are two options. The first one is to transfer data to a single database and mine it on that database. The second option is to mine them independently and still generate association rules for the combination of the data in multiple

databases.[1]. A large majority of organizations have computerized all or a part of their daily activities. Let us consider a company, which has several branches in different locations with each branch having its own database. The main branch or top level within the organizational hierarchy is responsible for development and decision making within the entire company. Let us consider the following multi-database environment shown in Figure 2.1.

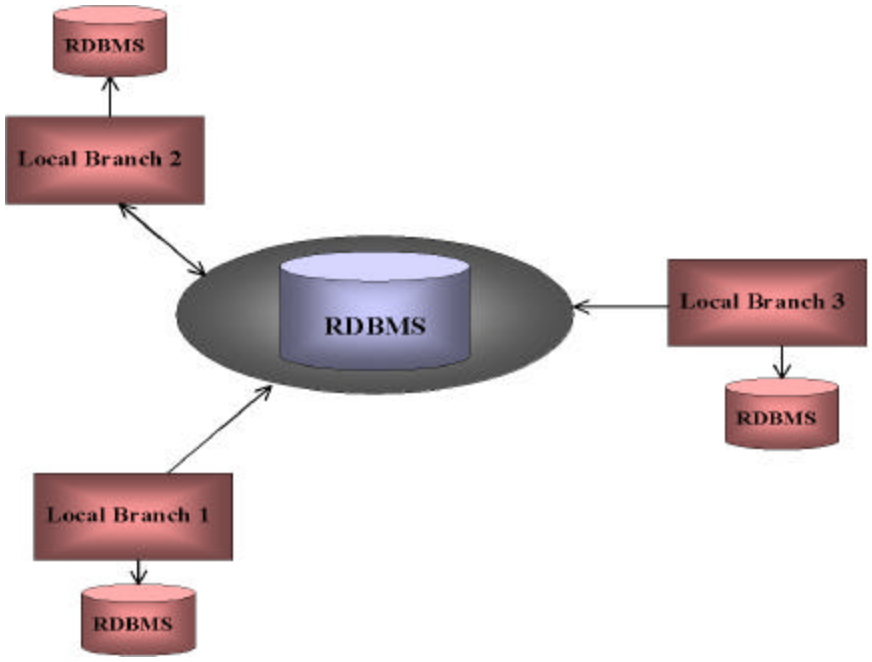


Figure 2.1 A Multi-Database Environment

The development of multi-database association rule mining is a challenging and critical task since it requires knowledge of all the data stored at different locations and the ability to combine partial results from individual RDBMS's into a single result. The individual databases have to be analyzed to generate rules to make local decisions. It would be easier for the organization to make decisions based on the rules generated by the individual branches, rather than using the raw data. If the raw data from each of the

individual databases were sent to a single database to generate the rules, certain useful rules, which would aid in making decisions about local branches, would be lost. For example a rule such as "50% of the branches in the north saw a 10% increase in the purchase of printers when Digital cameras and memory cards were purchased together" would not be generated if the raw data was transferred. If the raw data from all the databases were transferred to a single database then each of the individual branches would not be generating the rules with respect to its data. In such a case the organization may miss out certain rules that were prominent in certain branches and were not found in the other branches similar to the above example. Generating such rules would aid in making decisions about specific branches.

[2] presents a weighting model for synthesizing high-frequency association rules from different sources. A high-frequency rule is the one that is supported by most of the data sources. High-frequency rules are preferred for two reasons. First, a company headquarter is interested in the rules supported by most of its branches for corporate profitability. Second, high frequency rules have larger chances to become valid rules in the union of all data sources than the low-frequency rules do. The proposed model assigns a high-weight to a data source that supports/votes more high-frequency rules and a lower weight to a data source that supports/votes less high-frequency rules. A relative synthesizing model using clustering is used when the data source is unknown (e.g., collected from the web, journals and books). This model is different form parallel and distributed mining and metalearning because they do not produce a global learning model from classifiers from different data sources.

Although a multi-relational database can be transformed into a single universal relation, practically this can lead to many issues such as universal relations of unmanageable

sizes, infiltration of uninteresting attributes, loss of useful relation names, unnecessary join operations, and inconvenience for distributed processing. [1] Discusses a new multi-database mining process. The patterns in multi-databases are divided into the following classes:

Local patterns: Local branches need to consider the original raw data in their datasets so they can identify local patterns for local decisions.

High-vote patterns: These are the patterns that are supported by most of the branches and are used for making global decisions.

Exceptional patterns: These patterns are strongly supported by only a few branches and are used to create policies for specific branches.

The mining strategy used in [1] identifies two types of patterns, high-vote patterns and exceptional patterns. The discovery of these patterns can capture certain distributions of local patterns and assist global decision-making within a large company.

CHAPTER 3

PARTITIONED APPROACH TO ASSOCIATION RULE MINING

Different approaches have been proposed to generate association rules effectively. These proposed approaches have their own advantages and disadvantages. In this chapter we will revisit the partition algorithm in finer detail. The outline of this chapter is as follows: Section 3.1 adapted and evaluated the performance of the partition algorithm to suit RDBMS. Section 3.2 discusses the two variants of the partition algorithm proposed in this thesis

3.1 Database Approach To Partition Algorithm

Most of the algorithms for discovering association rules [5, 6, 15] require multiple passes over the database. The database was read completely for each pass resulting in a large number of disk reads in the case of disk resident databases placing a huge burden on the I/O subsystem. Network congestion problems and poor resource utilization was common in cases where data was to be retrieved from a central database server over a network. The partition algorithm [12] is an efficient algorithm for mining association rules in large databases. In this section we present the performance of the partition algorithm for multiple databases.

SQL operations over Relational databases were used. There was no change in Phase I of the algorithm. Each database was considered an individual partition and the frequent

itemsets were generated for each of the databases. In Phase II of the algorithm the frequent itemsets from each of the partitions were merged to form two sets of itemsets. The first set is the global frequent itemsets, which correspond to itemsets that are large in all the partitions (databases). The second set is the set of global candidate itemsets, which is the union of all the frequent itemsets from each of the partitions (and does not include the global frequent itemsets).

A TIDLIST is created for the entire database. This would incur the shipping cost of the partitions to one database. As the data is assumed to be distributed over different databases, they need to be shipped to a single database to combine and create the TIDLIST. The TIDLIST was used for counting the support of the itemsets in the global candidate itemsets and the itemsets satisfying the user specified support were added to the set of global frequent itemsets.

3.1.1 Methodology for Experiments

The performance results presented in this thesis are based on datasets generated synthetically using IBM's data-generator. The nomenclature of these datasets is of the form "TxxIyyDzzzK", where "xx" denotes the average number of items present per transaction, "yy" denotes the average support of each item in the dataset and "zzzK" denotes the total number of transactions in "K"(1000s). The experiments have been performed on Oracle 8i and IBM DB2 / UDB V7.2 (installed on a machine running Microsoft Windows 2000 Server with 512MB of RAM). Each experiment has been performed 4 times. The values from the first run are ignored so as to avoid the effect of the previous experiments 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, we have found that the percentage difference of each run with respect to the average is less than one percent. 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 descriptions is relatively insignificant, they are not reported. For the purpose of reporting experimental results in this thesis, we have shown the results only for three datasets – T5I2D500K, T5I2D1000K and T10I4D100K for most of the optimizations.

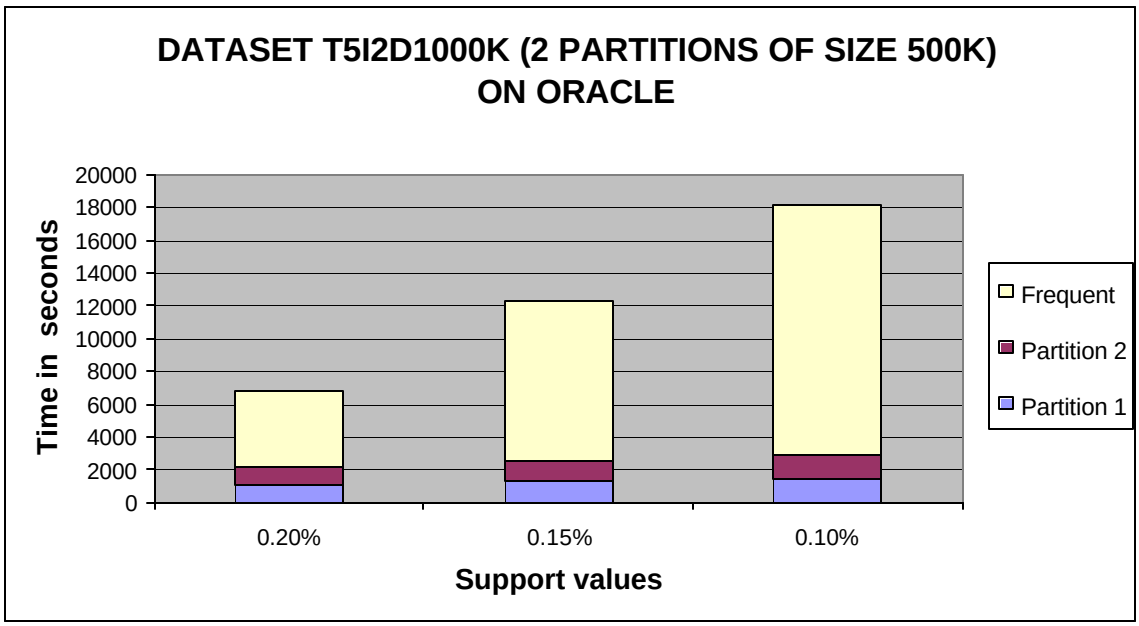


Figure 3.1 Performance Of TIDLIST Approach On T5I2D1000K Dataset

Figure 3.1 shows the performance of the TIDLIST approach for a T5I2D1000K dataset. The dataset is divided into two equal partitions each of size 500K. The analysis of the time taken for the different phases shows that the Phase II is the most time consuming. In Phase II, the TIDLIST is created for the whole dataset.

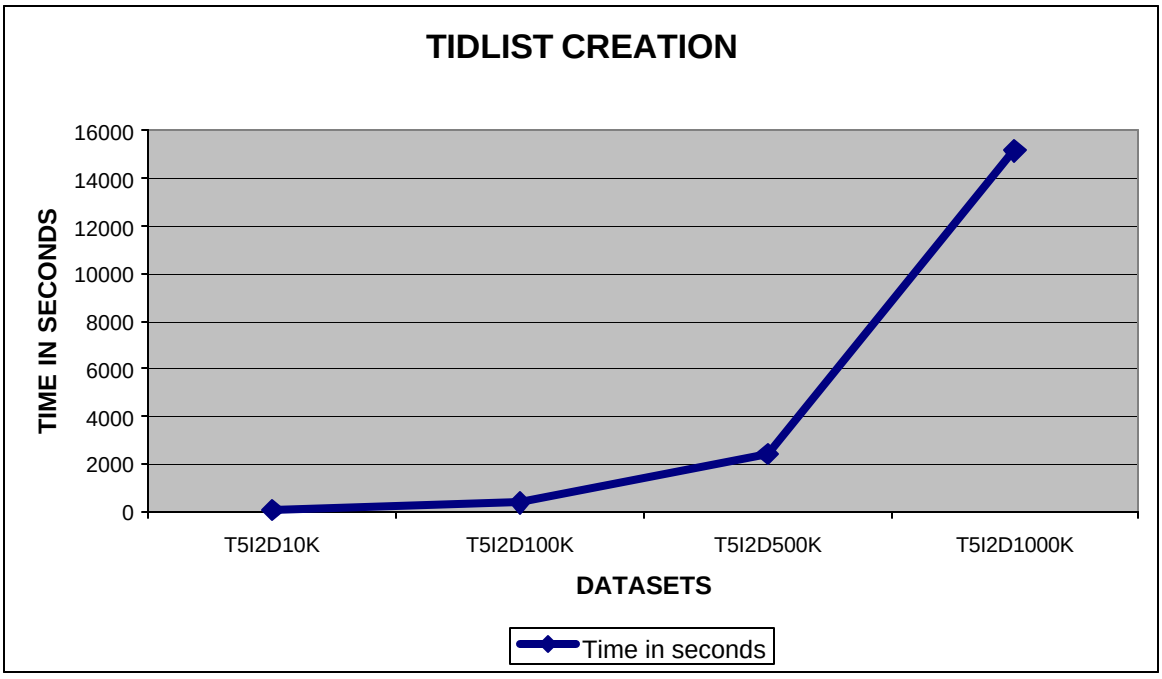


Figure 3.2 Time Taken For TIDLIST Creation For Different Datasets

Figure 3.2 shows the time taken for the TIDLIST creation for datasets of different size. The TIDLIST creation time increases exponentially as the size of the dataset increases. The partitioned approach although seems to work well for main memory databases, its performance for partitioned databases is not acceptable. The creation of the TIDLIST and the shipping of the partitions to a single database need to be avoided. In the next section, two

approaches have been proposed to overcome the above inefficiency of the partition algorithm for multiple databases.

3.2 Proposed Extensions To Partition Algorithm

This section discusses two approaches Approach I and Approach II that have been proposed for the partition algorithm useful for multiple databases. The following notation is used in the remainder of the thesis.

Table 3.1 Notations Used For Partitioned Approach

Notation	Meaning
C^P_K	***Local Candidate Itemsets:*** Set of local candidate k-itemsets in partition P.
F^P_K	***Local Frequent Itemsets:*** Set of local frequent k-itemsets in partition P.
C^G_K	***Global Candidate Itemsets:*** Set of global candidate k-itemsets.
F^G_K	***Global Frequent Itemsets:*** Set of global frequent k-itemsets.
$NBd(F^P_K)$	***Negative Border:*** Set of local non-frequent k-itemsets in partition P.

The negative border of frequent k-itemsets corresponds to those itemsets that did not satisfy the support in pass k. That is $NBd(F^P_K) = C^P_K - F^P_K$. Given a collection $F \subseteq P(R)$ of

sets, closed with respect to set inclusion relation, the NBd(F) of F consists of the minimal itemsets $X \subseteq R$ not in F.

3.2.1 Approach I

In the TIDLIST approach, the TIDLIST was created as a CLOB. In Approach I TIDLIST is not at all created and the k-way join approach is used instead. Some of the k-way join optimizations reported in [21] have been used. The two k-way join optimizations used are: Second-pass Optimization (SPO) and Reuse of Item Combinations (RIC). In a multiple database scenario, each of the individual databases is considered as a partition and the merging is done by choosing one of the databases. The changes made to the partition algorithm are described below.

3.2.1.1 Phase I

In this phase the frequent itemsets F^P_K are generated for each of the partitions. Along with the frequent itemsets in each of the partitions, the negative border of the frequent 2-itemsets $NBd(F^P_2)$ is also retained. These itemsets are used for counting the support in the Phase II of the algorithm. Only the negative border of the 2-itemsets is retained because when the second pass optimization is used, the generation of the 2-itemsets is the first step in each partition. Since the 2-itmeset generation is the first pass, there is no loss of information and the negative border of the 2-itemsets will have all possible 2-itemsets, which did not satisfy the support.

The other optimization for the k-way join -- Prune the Input table (PI) -- was not used during the implementation even though the combination of all the optimizations yielded better performance. This was because in the pruned input optimization, the input table would

be pruned by elimination all the records of those single itemsets whose support was less than the user specified support value. If this were done then the negative border of the 2-itemsets would not contain all the possible non-frequent 2-itemsets. After the frequent itemsets from all the partitions (databases) are generated, the frequent itemsets and the negative border of the frequent 2-itemsets from all the partitions are shipped to one database to do the remaining computation. This step is shown as an edge with label "1" in Figure 3.3. Merging the frequent itemsets from all the partitions generates the global candidate itemsets C^G_1, C^G_2, …, C^G_K.

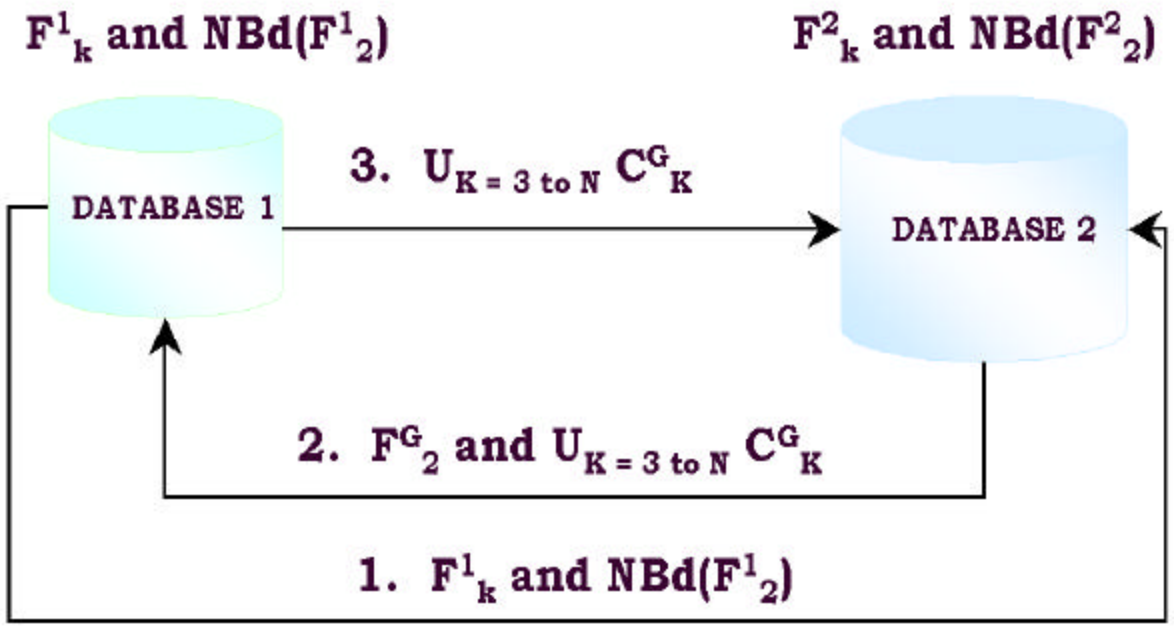


Figure 3.3 Data Transfer Using Approach I

3.2.1.2 Phase II

In this phase, the global frequent itemsets -- itemsets that are large in all the partitions – are generated. Merging the count obtained from the negative border and the frequent 2-itemsets from all the partitions generates the count for the remaining 2-itemsets in C^G_2. The itemsets satisfying the support are added to F^G_2. F^G_2 and $\cup_{k=3 \text{ to } n} C^G_K$ are shipped to all the databases to generate the counts of the remaining candidate itemsets. This is shown as an edge with label "2" in Figure 3.3.

Each of the databases generates a materialized table from the global frequent 2-itemsets using the Reuse of item combination optimization. The materialized table is used in the successive passes to generate the counts of the itemsets in the global candidate itemsets. Once the counts are generated in all the partitions they are shipped back to one database to do the final counting. This is shown as an edge with label "3" in Figure 3.3.

Figure 3.3 shows the data transferred in each of the steps. Database 1 and Database 2 are considered the 2 partitions. Database 2 is chosen for merging the frequent itemsets from all the partitions to global candidate itemset and for generating the final cumulative count of all frequent itemsets obtained from all the partitions in step "3".

3.2.1.3 Performance Analysis

Performance experiments were done on datasets of different sizes. Each data set was divided into 2 or 3 non-overlapping partitions. Figure 3.4 shows the performance of a T5I2D1000K dataset divided into 2 equal sized partitions each of size 500K. It is seen from the graph that the improvement in performance of Approach I compared to TIDLIST approach is 58% for a support values of 0.20% and the improvement increases to about 78%

for a support value of 0.10%. As the support value decreases the percentage improvement in the performance increases.

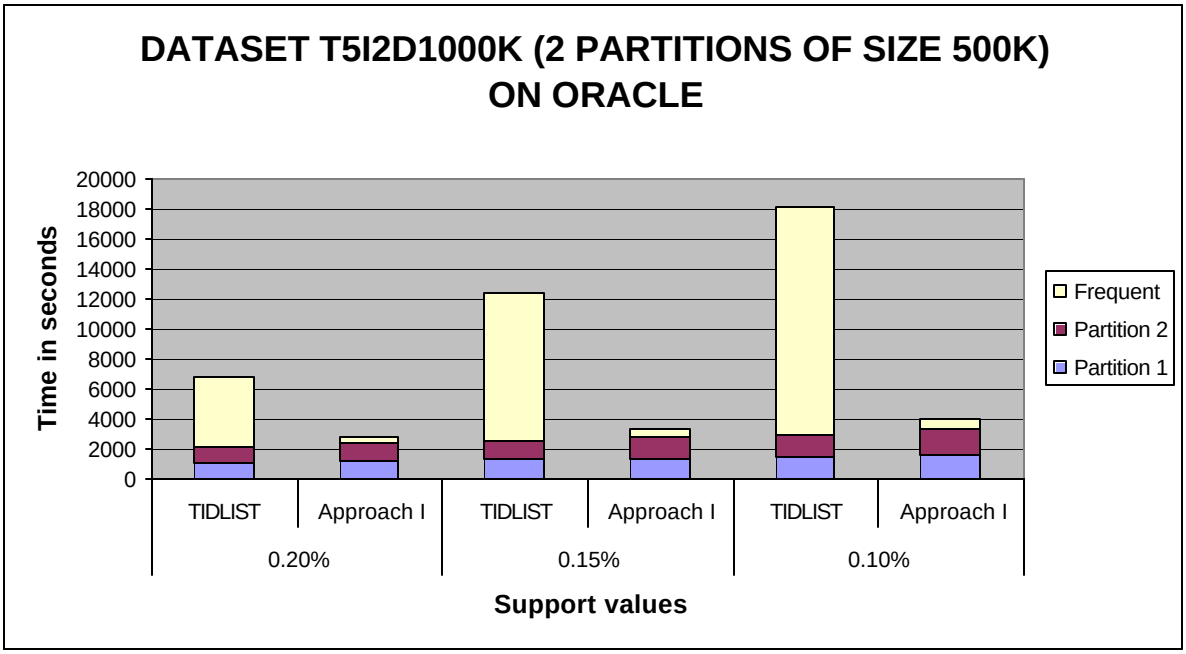


Figure 3.4 Performance Comparison Of TIDLIST And Approach I

Figure 3.5 shows the performance on Oracle and DB2 for a T5I2D500K dataset divided into 2 partitions of size 250K each. DB2 experiments did not complete for support values of 0.15% or lower even after running for 10-12 hours. For Oracle the improvement in performance was 35% for 0.20% support and it increased to 61% as support decreased to 0.10%. In DB2 the improvement in performance decreased from 82% to 44% as the support value decreased from 0.30% to 0.20%.

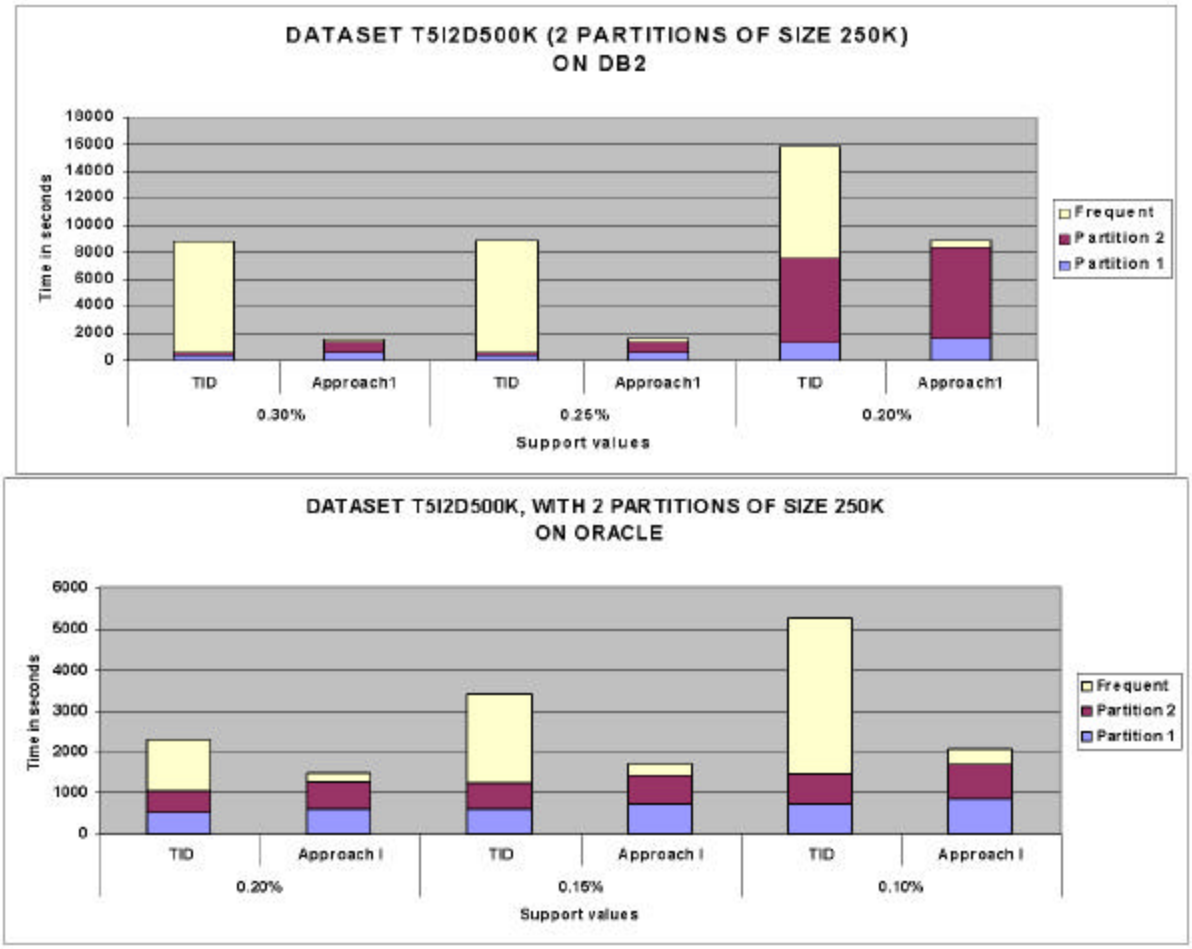


Figure 3.5 Performance Of TIDLIST And Approach I For T5I2D500K

Figure 3.6 shows the data transfer when there are 3 partitions (databases). At the end of each phase the intermediate results are transferred to one of the partitions to do the remaining computations.

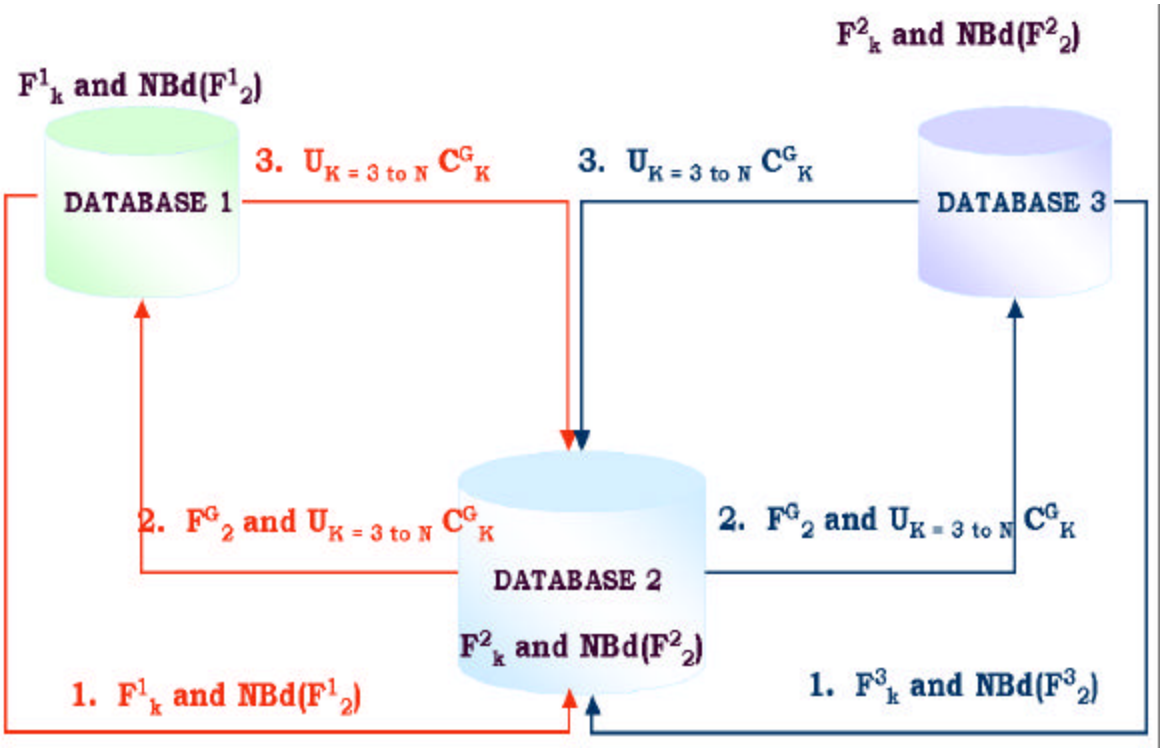


Figure 3.6 Data Transfer Using Approach I For 3 Partitions

The performance for T5I2D500K is shown in Figure 3.7. The dataset is divided into 3 partitions of size 200K, 200K and 100K. The performance is shown for Oracle and DB2. For DB2 the percentage improvement in performance decreases from 80% to 53% as the support value decreases from 0.30% to 0.20%. The performance in Oracle shows an increase from 18% to 75% as the support value decreases from 0.20% to 0.10%.

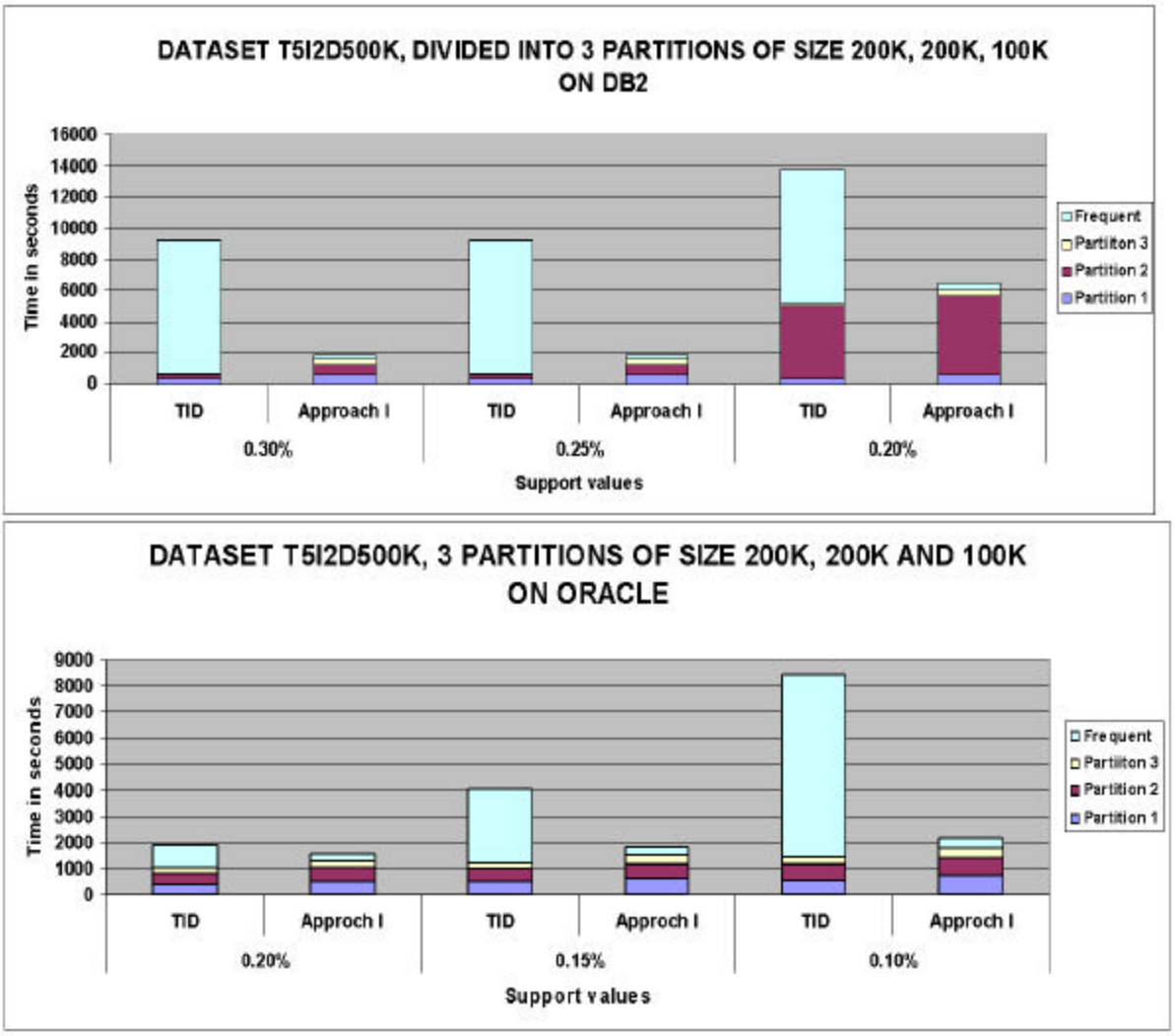


Figure 3.7 Performance Comparison Of T5I2D500K For TIDLIST And Approach I

3.2.1.4 Data Transfer

Table 3.2 shows the number of records transferred between the databases in each step. The input data denotes the transactional data. It is assumed that the dataset is divided into 2 equal sized partitions. The numbers in the Table 3.2 indicate the number of records transferred. For example, for the T5I2D10K dataset the input data has 27000 records and the

total records transferred using approach I between the two databases is 51845. It is observed that transferring the intermediate results is better for the datasets, which have more than 100K transactions.

Table 3.2 Data Transferred Using Approach I

Dataset	**Input data records**	**Step 1** $[F^1_K + NBd(F^1_2)]$ **records**	**Step 2** $[F^G_2 + U_{K=3 \text{ to } N} C^G_K]$ **records**	**Step 3** $U_{K=3 \text{ to } N} C^G_K$ **records**	**Total records**
T5I2D10K	27000	50360	1393	92	51845
T5I2D100K	273000	199455	678	101	200234
T5I2D500K	1368500	319677	638	76	320391
T5I2D1000K	2736000	356696	632	75	357403

In Approach I only the negative border of the frequent 2-itemsets were retained in all the partitions. In the Phase II, a materialized table was created to do the support counting. The time taken to create a materialized table increases as the size of the dataset increases. In this approach the data is transferred 3 times between the partitions. Approach II was proposed to overcome the above drawbacks.

3.2.2 Approach II

In the Phase I of this approach the negative border of all the frequent itemsets in each of the partitions were retained as compared to the previous approach where only the negative border of the frequent 2-itemsets were retained. When the frequent itemsets are generated the data is transferred to one of the partitions to form the global candidate itemsets and the global

frequent itemsets. The global frequent k-itemsets are generated by merging the counts of the frequent k-itemsets and the negative border of the frequent k-itemsets. Figure 3.8 shows the data transfer in Approach II.

Approach II is different from Approach I with regard to the number of times data is transferred between the databases and the itemsets that are retained. In Approach I only the negative border of the frequent 2-itemsets is retained. Since retaining the negative border does not require any additional computation, in Approach II, the negative border of all the frequent itemsets are retained for all the databases. In Phase II of Approach I, the global frequent 2-itemsets are generated using the local frequent 2-itemsets and their negative border from all the databases. The results have to transferred to the individual databases to generate the remaining (3-k) – itemsets, which requires the scanning the input data in each of the databases to generate the counts. But in Approach II, all the global frequent itemsets are generated using the local frequent itemsets and their negative border from all the databases. An additional scan of the database is not required and the intermediate results are transferred only once as compared to 3 times in Approach I.

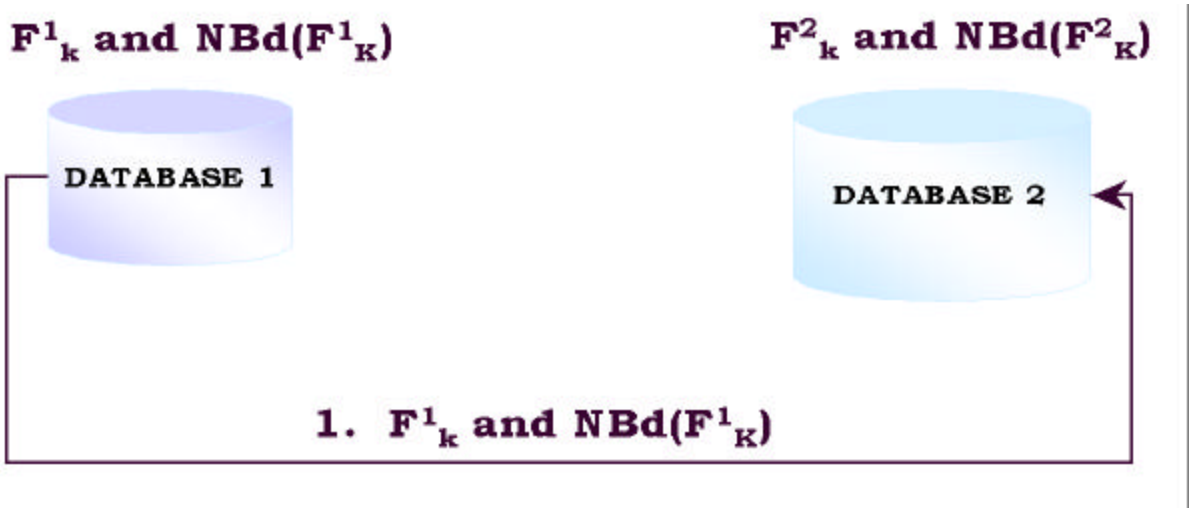


Figure 3.8 Data Transfer In Approach II

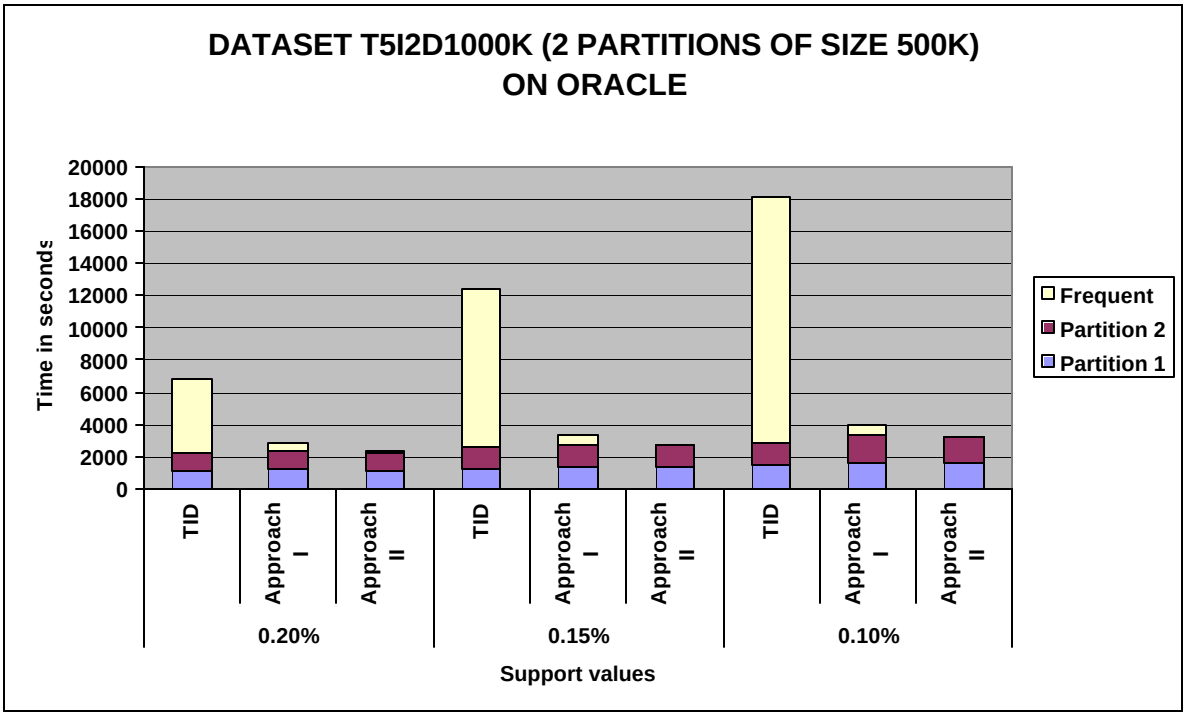


Figure 3.9 Performance Comparison Of All The Approaches With 2 Partitions

The Figure 3.9 shows the performance comparison of the TIDLIST, Approach I and Approach II. A T5I2D1000K dataset was divided into 2 partitions of size 500K each. From the graph it is noted that the performance improved from 16% to 18% as the support value decreased from 0.20% to 0.10%.

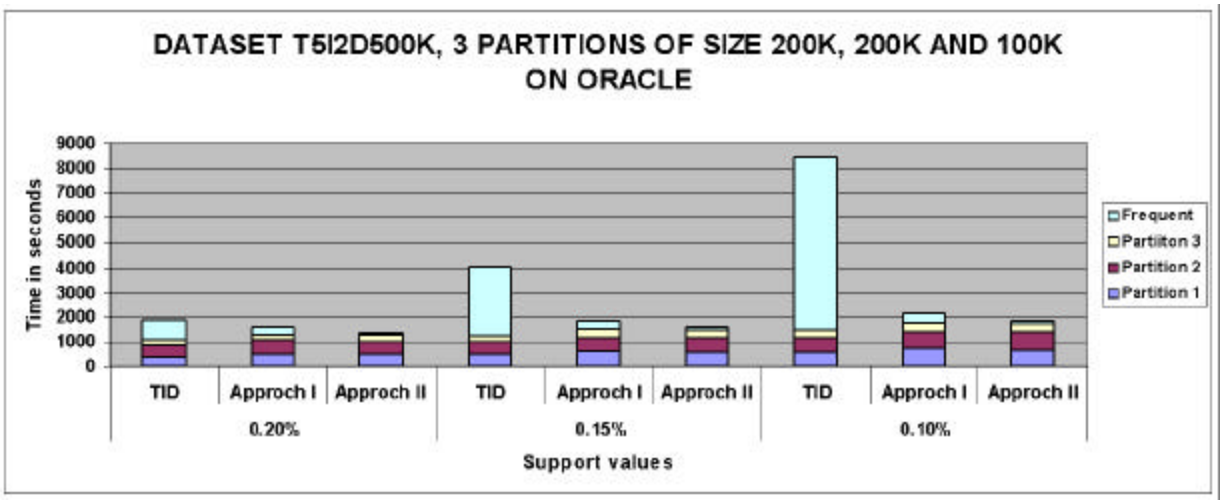


Figure 3.10 Performance Comparison Of All The Approaches With 3 Partitions

Figure 3.10 shows the performance comparison for all the 3 approaches on a T5I2D500K dataset divided into 3 partitions. For Oracle the performance improves from 14% to 16% as the support value decreases from 0.20% to 0.10%. Table 3.3 shows the number of records transferred between the partitions. For the datasets T5I2D100K and above transferring the intermediate relations would be better.

Table 3.3 Data Transfer For Approach II

Dataset	Input data records	Step 1 $[F^1_K + NBd(F^1_K)]$ records	Approach II
T5I2D10K	27000	50481	50481
T5I2D100K	273000	199521	199521
T5I2D500K	1368500	319740	319740
T5I2D1000K	2736000	356761	356761

Table 3.4 compares the number of records transferred in the case of Approach I and Approach II. It is noted that the data transfer in Approach II is slightly less for all the datasets. The data transferred in both the approaches showed a slight difference only because the frequent 2-itemsets and their negative border constitute a large number of records and they were transferred in both the approaches. In Approach I the global candidate (3-k)-itemsets and the global frequent (3-k)-itemsets were transferred which did not comprise a large number of records when compared to Approach II where the frequent (3-k)–itemsets and their negative border were transferred.

Table 3.4 Comparison Of Data Transfer For Approach I And Approach II

Dataset	Input data records	Approach I	Approach II
T5I2D10K	27000	51845	50481
T5I2D100K	273000	200234	199521
T5I2D500K	1368500	320391	319740
T5I2D1000K	2736000	357403	356761

It was noted that Approach II performed better than TIDLIST and Approach I for almost all the cases. This was because the creation of materialized table was eliminated and retaining the negative border does not require any additional computation. However there is a tradeoff associated with the Approach II. This approach may miss out the count of itemsets, which are globally large but locally small in a few partitions. The count of some k-itemsets whose subset did not appear either in the frequent itemsets or its negative border in the earlier passes may be misses. Figure 3.11 shows the error observed in the frequent itemsets generated. The frequent itemsets generated using the TIDLIST approach and Approach I was compared with the itemsets generated in Approach II. Approach II showed some error in the number of frequent itemsets generated in each pass. It was seen that there was some error for the smaller datasets with lower support values. No error was noted for datasets T5I2D100K and above.

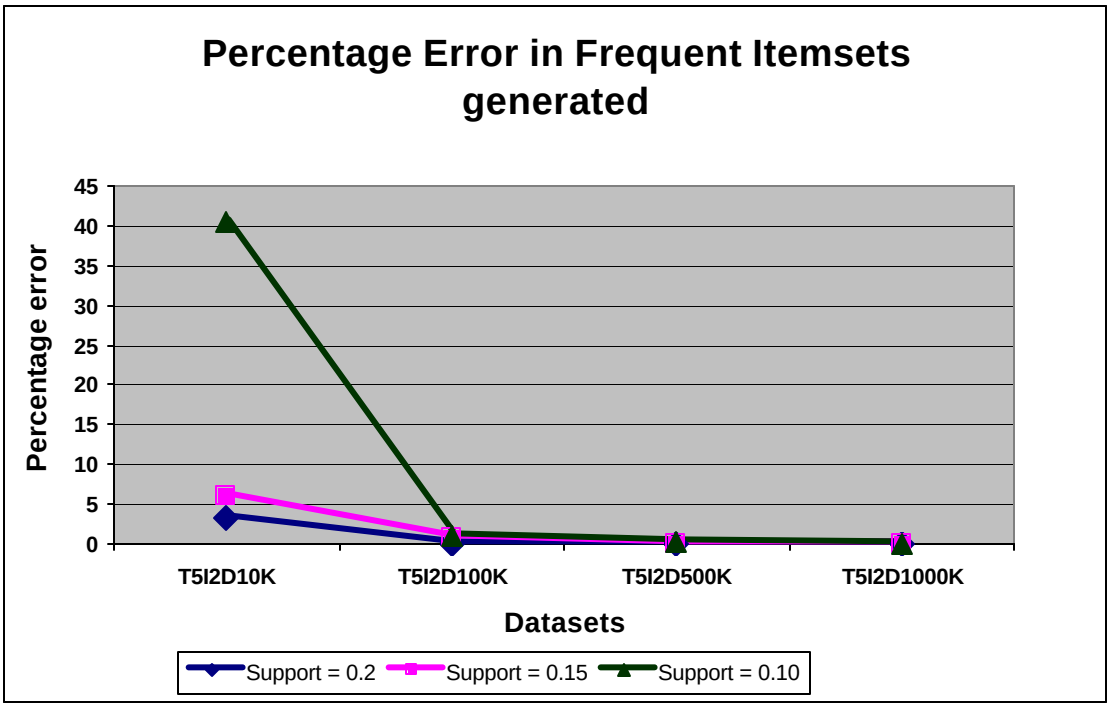


Figure 3.11 Error Analysis

Figure 3.12 shows the performance comparison between Approach I and Approach II on Oracle for different support values and datasets. The performance improvement in generating the rules between Approach I and Approach II were compared. It is noted that Approach II performs 14% to 18% better compared to Approach I. The reason being in Approach II the negative border is maintained for all the frequent itemsets and the itemsets are transferred only once. But in Approach I, though the negative border is retained for only the frequent 2-itemsets, in the phase II a materialized table is created for each of the databases to count the support of the itemsets in the global candidate itemsets. The time it takes to create the materialized table increases as the size of the dataset increases as a result of which there is an improvement in the performance between both the approaches.

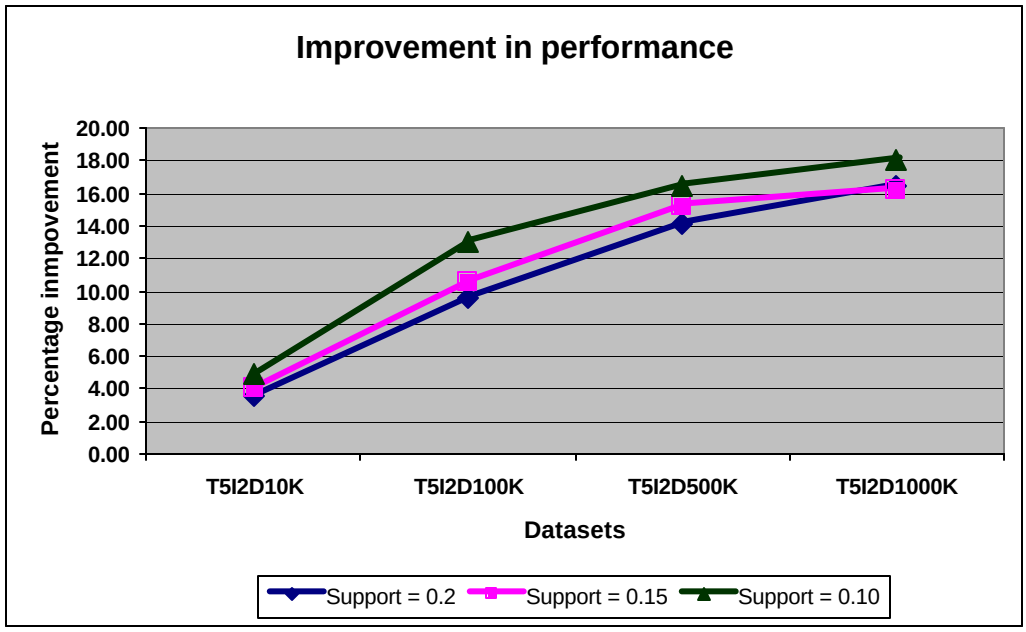


Figure 3.12 Comparing performance of Approach I and II on Oracle

Based on the experiments performed using the partitioned approach we can summarize the effect of the optimizations as follows:

The TIDLIST approach has its drawbacks when it is implemented in a database approach because of the use of CLOBs for TIDLIST creation, which is a very time consuming operation. The two approaches were proposed to optimize the TIDLIST approach to gain better performance that may useful for multiple databases.

CHAPTER 4

INCREMENTAL ASSOCIATION RULE MINING

In CHAPTER 3 we discussed how the partition algorithm could be used to mine data from multiple sources. In this chapter we will discuss the incremental addition of data and the generation of association rules without performing recomputation. Section 4.1 discusses the incremental mining algorithm for updating frequent itemsets. In Section 4.2 various performance experiments performed on Oracle and DB2 and their results are presented.

4.1 Incremental Updation Of Frequent Itemsets

Data mining is used for pattern discovery and query resolution in large data repositories. The rules generated in this way reflect the current state of the database. Techniques have to be developed to handle large volumes of data and maintain rules over significantly long periods of time. Updates to the transaction database may invalidate the existing rules or introduce new rules. The rule generation is a straightforward process and computationally inexpensive. Hence, it is not critical to develop an incremental rule generation algorithm. A naïve solution to the update problem is the recomputation of the frequent itemsets for the updated database. This is an inefficient way because all the

computations done initially are wasted. The ideal way would be to develop an incremental algorithm so that the computation effort spent on the original data is effectively utilized.

[8] has proposed an algorithm for incrementally updating the frequent itemsets. An existing database may be updated when new transactions are added or existing transactions are removed from the database. When new transactions are added to the database, an old frequent itemset could potentially become infrequent in the updated database or an old infrequent itemset could potentially become frequent in the new database. The proposed algorithm finds the new frequent itemsets with minimal re-computation when transactions are added to or deleted from the existing database. The following notation is used for describing the incremental approach.

Table 4.1 Notations Used In Incremental Approach

Notation	Meaning
DB	Transactions in original database
db	Transactions that are newly added
DB+	Transactions in the updated database (**DB** È **db)**
$\mathbf{F^{DB}, F^{db}, F^{DB+}}$	Frequent itemsets in the respective databases
$\mathbf{NBd(F^{DB}), NBd(F^{db}), NBd(F^{DB+})}$	Negative borders for the respective frequent itemsets.

The two important characteristics of the algorithm are: It makes use of the negative border concept. Along with the frequent itemsets, the negative border is also maintained. The negative border consists of all itemsets that were candidates, but lacked the minimum support. For example in pass k, $NBd(F_k) = C_k - F_k$ where C_k is the set of candidate k-itemsets,

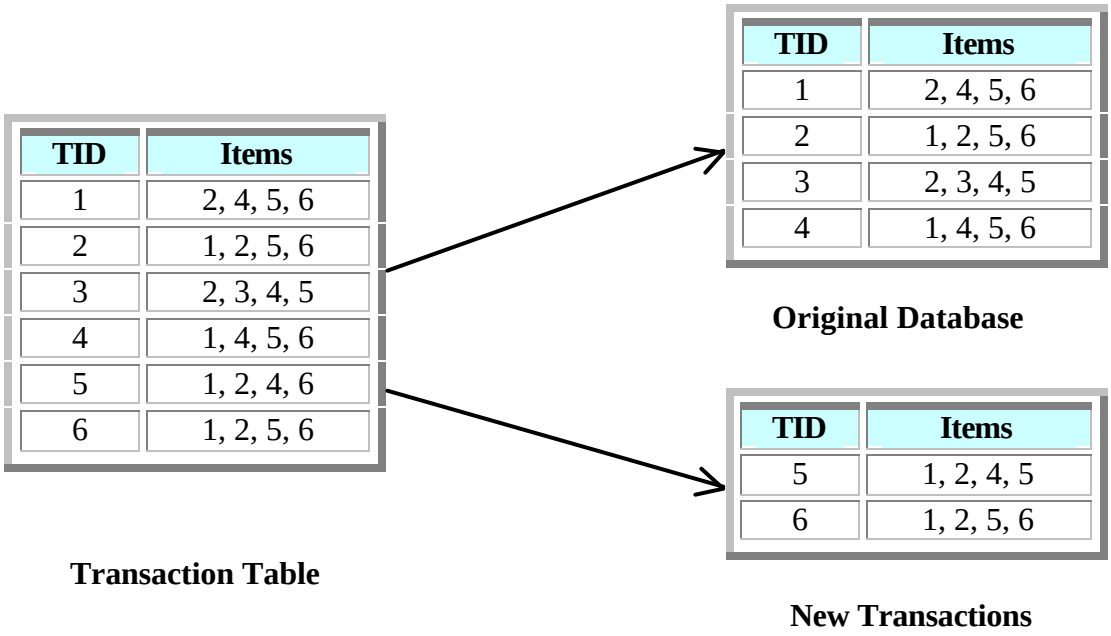
F_k is the set of frequent k-itemsets and NBd(F_k) is the set of k-itemsets in the negative border. The frequent itemsets for the increment database are computed. A full scan of the database is required only if the negative border expands, that is, if an itemset outside the negative border gets added to the frequent itemsets or its negative border. Figure 4.1 shows the incremental mining algorithm for updating the frequent itemsets.

```
function Update-Frequent-Itemset(F^DB, NBd(F^DB),db)

//DB and db denote the number of transactions in the original
database and the increment database respectively.
1 Compute F^db
2 for each itemset s ∈ F^DB ∪ NBd(F^DB) do
     t_db(s) = number of transactions in db containing s
     F^DB+ = ϕ
3 for each itemset s ∈ F^DB do
     if (t^DB(s) + t^db(s)) ≥ minsup * (DB + db)
          then F^DB+ = F^DB+ ∪ s
4 for each itemset s ∈ F^db do
     if s ∉ F^DB and s ∈ NBd(F^DB) and (t^DB(s) + t^db (s)) ≥
     minsup * (DB + db)
     then F^DB+ = F^DB+ ∪ s
5 if F^DB ≠ F^DB+ then
     NBd(F^DB+) = negativeborder-gen(F^DB+)
else NBd(F^DB+) = NBd(F^DB)
6 if F^DB ∪ NBd(F^DB) ≠ F^DB+ ∪ NBd(F^DB+) then
     S = F^DB+
repeat
     compute S = S ∪ NBd(S)
until S does not grow
F^DB+ = {x ∈ S | support(x) ≥ minsup}
//support(x) is the support count of x in DB ∪ db
NBd(F^DB+) = negativeborder-gen(F^DB+)
```

Figure 4.1 Incremental Mining Algorithm

Figure 4.2 shows a transaction table (DB^{+}). The original database (DB) has 4 transactions and the new database (db) has 2 transactions.



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

Transaction Table

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

Original Database

TID	Items
5	1, 2, 4, 5
6	1, 2, 5, 6

New Transactions

Figure 4.2 Updated Database

Initially the original transactions are mined to generate the frequent itemsets (F^{DB}) for the user specified support. While the frequent itemsets are generated, their negative border $NBd(F^{DB})$ is also generated and retained. The negative border is later used to avoid recomputation when new transactions are added to the database. The frequent itemsets and their negative border for DB are shown in Figure 4.3.

Frequent Itemsets

Itemset	Count
{ 1, 5 }	2
{ 1, 6 }	2
{ 2, 4 }	2
{ 2, 5 }	3
{ 2, 6 }	2
{ 4, 5 }	3
{ 4, 6 }	2
{ 5, 6 }	3

Negative Border

Itemset	Count
{ 1, 2 }	1
{ 1, 4 }	1

Itemset	Count
{ 1, 5, 6 }	2
{ 2, 4, 5 }	2
{ 2, 5, 6 }	2
{ 4, 5, 6 }	2

Itemset	Count
{ 2, 4, 6 }	1

Figure 4.3 Frequent Itemsets And Negative Border In DB

When new transactions are added to the database (db) the frequent itemsets for the new transactions F^{db} are generated for the same user specified support value as shown by step 1 in Figure 4.1. The count for each of the itemsets in $F^{DB} \cup NBd(F^{DB})$ is obtained from the new transaction database (db) as shown by step 2 in Figure 4.1. This is shown in Figure 4.4.

Frequent Itemsets

Itemset	Count
{ 1, 2 }	2
{ 1, 4 }	1
{ 1, 5 }	1
{ 1, 6 }	2
{ 2, 4 }	1
{ 2, 5 }	1
{ 2, 6 }	2
{ 4, 6 }	1
{ 5, 6 }	1

Itemset	Count
{ 1, 2, 4 }	1
{ 1, 2, 5 }	1
{ 1, 2, 6 }	2
{ 1, 4, 6 }	1
{ 1, 5, 6 }	1
{ 2, 4, 6 }	1
{ 2, 5, 6 }	1

Itemset	Count
{ 1, 2, 4, 6 }	1
{ 1, 2, 5, 6 }	1

Figure 4.4 Frequent Itemsets In The New Transactions

The following conditions to be checked to perform the update of the frequent itemsets.

4.1.1.1 Case 1

The support is generated for each itemset in F^{DB} from F^{db} and the itemset is added to F^{DB+} if support is satisfied. This is shown by step 3 in Figure 4.1. This would result in 2 cases. First, some itemsets, which were large in F^{DB} may remain large even after adding the count from F^{db} as shown by itemset {1,5,6} in Figure 4.5. Second, some itemsets, which were large in F^{DB} may turn out to be small when added with the count in F^{db} and it may move from the frequent itemsets to the negative border as shown by itemset {4,5,6} in Figure 4.5.

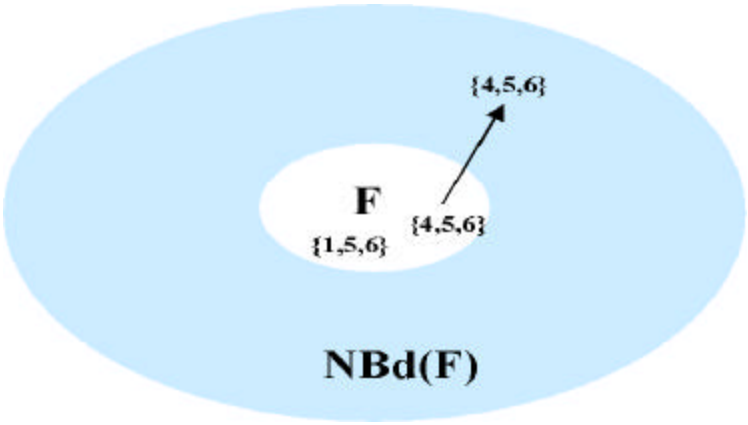


Figure 4.5 Case 1 For Incrementally Updating Frequent Itemsets

4.1.1.2 Case 2

The support of all the itemset that is common to the itemsets in F^{db} and NBd(F^{DB}) are counted and added to F^{DB+} if the support is satisfied. This case is depicted by step 4 in Figure 4.1. There are 2 situations that can arise here as shown in Figure 4.6.

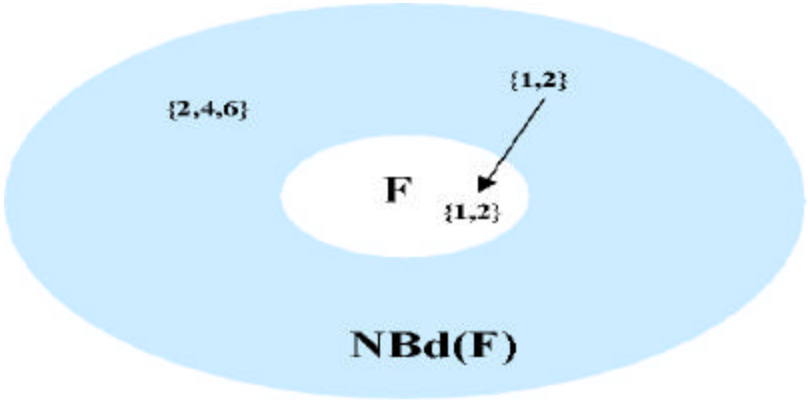


Figure 4.6 Case 2 For Incrementally Updating Frequent Itemsets

The first situation that arises is that some itemset which was in NBd(F^{DB}) may not satisfy the support and it would remain in NBd(F^{DB}) as shown by the itemset {2,4,6} in Figure 4.6. The second situation would be when an itemset that was in the NBd(F^{DB}) would become frequent as shown by itemset {1,2} in Figure 4.6.

4.1.1.3 Case 3

The support of all the itemsets that are in F^{db} and not present in F^{DB} or NBd(F^{DB}) are counted. There are two situations possible in this case. In the first case an itemset {1,2,6} in F^{db} would satisfy the support and be added to F^{DB+}. The steps 3 and 4 in Figure 4.1 would not take into account the count of such itemsets because all the subsets of {1,2,6} were not frequent as shown in Figure 4.7. The second situation would be an itemset {11,12} in F^{db} which did not have any of its subsets in F^{DB} or NBd(F^{DB}) as shown in Figure 4.7.

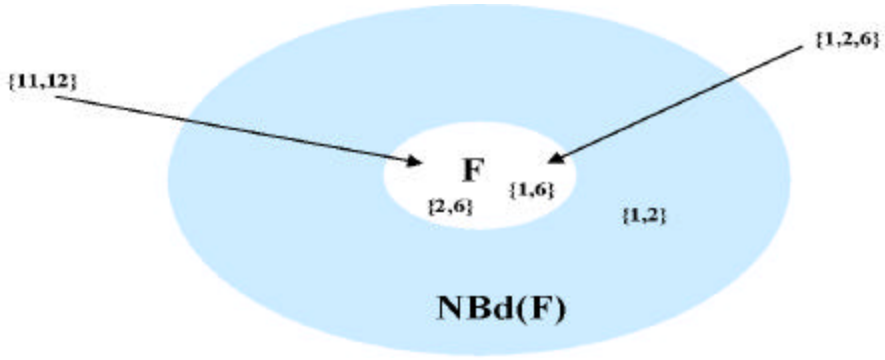


Figure 4.7 Case 3 For Incrementally Updating Frequent Itemsets

This was done by step 6 of the algorithm shown in Figure 4.1. The input relation would be scanned multiple times to generate the count of all such itemsets. This was done by

repeating the join and the prune step in the frequent itemset generation. Candidate itemsets were generated in each pass from which the frequent itemsets were generated. However those itemsets, which were totally new and did not have any of its subsets in F^{DB} or $NBd(F^{DB})$ were not counted in this step.

In our approach we have added the following step to take care of Case 3 discussed above.

```
for each itemset s ∈ F^db do
    if s ∉ F^DB and s ∉ NBd(F^DB) and (t^DB(s) + t^db (s)) ≥
    minsup * (DB + db) then F^DB+ = F^DB+ ∪ s
```

Here the original database DB is used to generate the count of all the itemsets that are present in F^{db} and not present in F^{DB} and $NBd(F^{DB})$. A materialized table [21] (RIC optimization of k-way join) was created in each pass to avoid redoing the same join operations in each pass. The steps 5 and 6 in Figure 4.1 denote the steps to expand the negative border. These steps require multiple passes over the transactional database; in our approach we are retaining the negative border for the itemsets in F^{db} also. The frequent itemsets and the negative border of the frequent itemsets were merged to generate the negative border for the update database. The negative border expanded this way has the same itemsets that are generated using the step 6 in Figure 4.1 because the negative border is the minimal collection of the non-frequent itemsets, which in closed with respect to the set inclusion operation.

4.2 Performance Evaluation

The performance results presented in this section are on datasets generated synthetically using the IBM's data-generator. The datasets used are the same as the ones used in the partition based approach. The experiments have been performed on Oracle 8i and IBM DB2 / UDB V7.2 (installed on Windows 2000 server with 512MB of RAM).

A percentage of the transactions of the database are considered as the original database and the remaining transactions are added incrementally in percentages. The experiments are performed for at least 3 increments to the original database as in most cases recomputing may turn out to be better than incremental mining during the initial iterations till the size of the dataset grows considerably. Two percentages of increments 5% and 10% were chosen. The graph in Figure 4.8 compares the time taken to generate rules by recomputing (RC) and by using the incremental mining algorithm (IM).

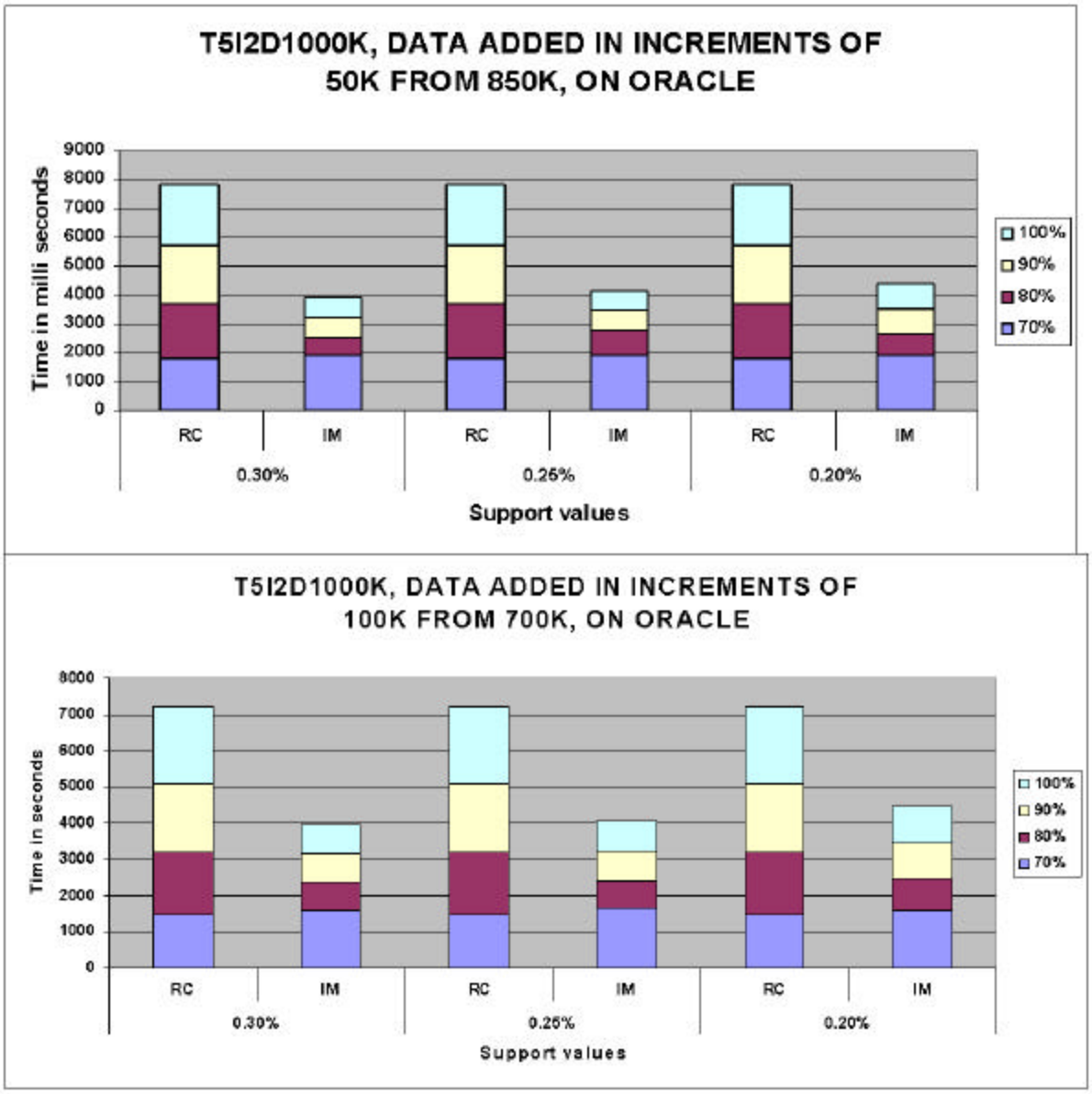


Figure 4.8 Performance For T5I2D1000K On Oracle

From the Figure 4.8 it is noted that the improvement in performance for increment sizes of 50K was about 50% for support of 0.30% and it decreased to 44% for a support value of 0.20%. With 100K increments, the percentage improvement in performance was 45% for support of 0.30% and it decreased to about 38% as the support value decreased to 0.20%.

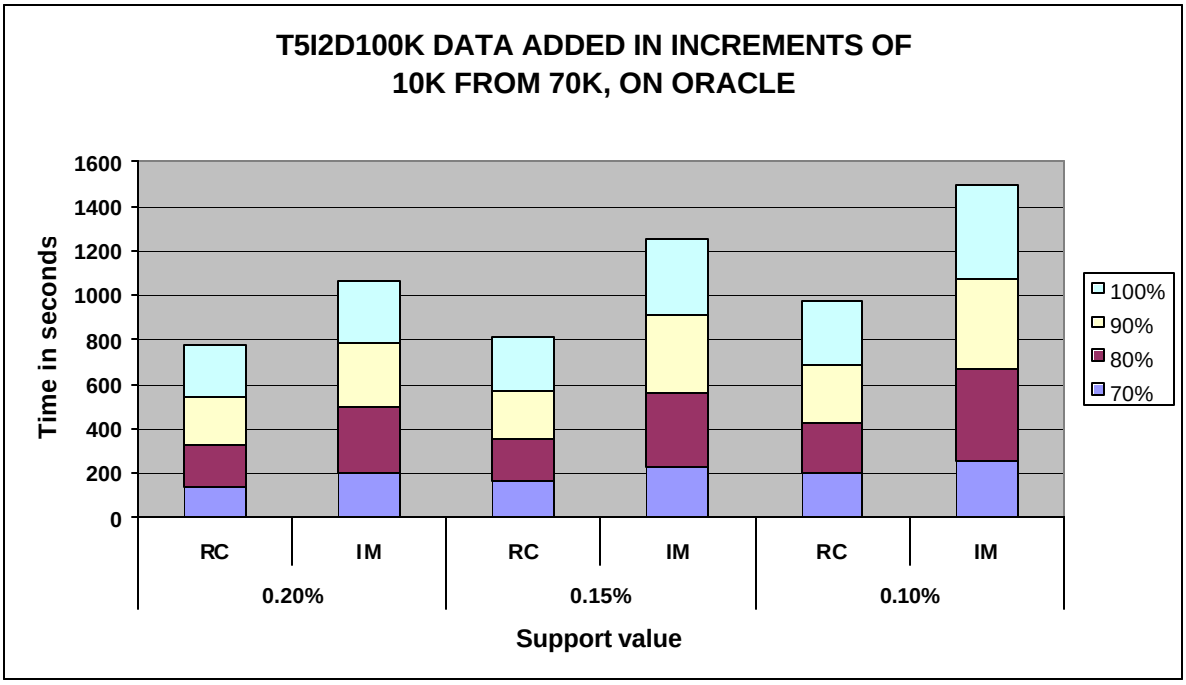


Figure 4.9 Performance Of T5I2D100K On Oracle

Figure 4.9 shows the performance comparison of the incremental algorithm with recomputation. It can be noted that recomputation performs better than incremental mining. This is because when the size of the increment is very small, saving the negative border information takes more time because for the initial passes, the number of candidates generated is large and there are more itemsets in the negative border. The large number of itemsets generated may be due to fact that for the smaller datasets almost all the itemsets generated satisfy the support value. The general trend observed is that for the smaller datasets with lower support values the number of passes and the itemsets generated in each pass is slightly higher when compared to the larger datasets. For example let us consider a T5I2D10K dataset with a support of 0.10%, with 7K transactions in the original database.

The support in terms of the number of transactions would be 7 in this case. When an increment with 1K transactions is added then the support in terms of the number of transactions would be 1. Almost all the candidate itemsets generated would satisfy a support of 1. Hence there is more computation involved for generating the frequent itemsets and the negative border of the frequent itemsets.

Figure 4.10 shows the performance comparison of the incremental mining algorithm with the recomputation. In the first graph there were 350K transactions in the original database and three increments each of size 50K were added. The improvement in performance compared to recomputation was 31% for a support value of 0.20% and it increased to about 36% as the support value increased to 0.30%. In the second graph the original database had 425K transactions and three increments each of size 25K were added. The improvement in performance was 35% for a support value of 0.20% and it increased to about 48% as the support value increased to 0.30%

Figure 4.10 Performance Of T5I2D500K On Oracle

Based on the experiments performed using the incremental approach we can summarize the effect of the optimizations as follows:

Incremental mining performs better when compared to recomputation for larger datasets. Retaining the negative border for the new transactions avoids the use of the updated database DB+ for generating the negative border when it expands.

CHAPTER 5

OTHER CONTRIBUTIONS

In the previous two chapters we discussed algorithms for generating the frequent itemsets, which is the first step in association rule mining. This chapter consists of other contributions that are required for the association rule generation and for performing large number of experiments. Section 5.1 explains the Configuration file that is used in running the mining tool in a batch mode. Section 5.2 discusses the Log files that are generated during the mining process and how these logs can be formatted to give us a better understanding of the results.

5.1 Configuration File

There are two ways for using this mining tool. The first is using the GUI and other is using the configuration file. Running this mining tool using GUI has been described in [20, 22]. The GUI is useful for a non-expert (or a novice), but needs some human intervention to provide the configuration needed for mining. The configuration file is useful for automating the mining process. It consists of a number of parameters, which once specified correctly, can be used for mining in an unattended mode. It can also be used for mining several datasets with varying mining configurations without any user intervention. The variables defined in the configuration file are:

RDBMS Name: The RDBMS name (Oracle or DB2) where the input relation is present.

Database Name: The database that contains your input relation.

UserId: The user who has access over the input relation.

Password: The password associated with the UserId – needed to connect to the database.

Log File: The name of the Result Log file to generate.

Approach Number: The approach number to be used for mining. It is an integer value. All the approaches and their optimizations are given a unique integer value to identify them.

Table Name: The name of the input relation.

Percentage: The percentage is used to select specific number of transactions for the partitioned and incremental approach.

Support: Minimum support value to be used for mining. This is in percentage.

Confidence: The confidence value to be used for rule generation. It is an integer value (as percentage)

Stop Level: The maximum number of passes to go before stopping.

Debug: If true, then prints the debug statements.

Skip Rules: If true, the program stops after the generation of frequent itemset. Rule generation is skipped.

Reverse Mapping: If true, the results (item ids) are mapped back to their original names.

Log Results to file: If true, trace values will be written to the Log File.

For each experiment, the values of all these variables are written in a single line in the order of the variables shown above and are demarcated by a "$" sign. Thus if the configuration file contains several such lines, the mining algorithms will be invoked that many times. To skip a line, the line should start with the word "REM". Below is an example of some mining configurations.

REM Experiment on DB2. Approach -Incremental

DB2$Sample$ntmining$ntmining$D_A23_T5I2D500K.txt23T5I2D500K$10$0.2999%$50$8$false$true$false$true

Here the first line is ignored as it starts from the word "REM". For second line values are used as follows:

RDBMS to use: DB2

Database Name: Sample

UserID: ntminig

Password: ntmining

Log File: D_A23_T5I2D500K.txt

Approach Name: 23 (for Incremental Mining)

Input Table: T5I2D500K.

Percentage: 10 (10% of transaction in input table constitute the increment)

Support: 0.2999 %

Confidence: 50 (percent)

Stop Level: 8

Debug: False (don't print debug statements)

Skip Rules: True (skip rule generation)

Reverse Mapping: False (don't do reverse mapping)

Log result to File: True (write the log file).

5.2 Writing Log File

Data mining is a time-consuming process and for certain mining configurations, mining a given dataset may take 10 to 15 hrs or even more. Since we have to compare the performances of these approaches with others, after a given time limit, if the approach does not complete, the mining process has to be killed. Also for the purpose of studying these algorithms, we need to know about their progress during mining a data set. Hence it is very important to note down the time at each step of the algorithm and produce a log file containing enough information. This log file can then be processed to generate useful information such as the number of passes completed, time taken for each pass, intermediate relations generated and cardinality of each of them, even if the mining process is killed before it completes. For this purpose, we generate two log files. One is the time log, which is written at the end of materialization of any relation generated during the mining process. This log (TimeLog) contains the time stamp of when a particular pass of the approach started and if any intermediate relations where generated, what is there cardinality. Below is a sample content of these logging files.

Contents of the TimeLog file:

Start-Approach 22. Table = T5I2D100K Support =99 Sat Oct 11 17:01:36 CDT 2003

// This indicates the start of Partitioned approach on input relation T5I2D100K.

// The support value, in terms of row count is 99.

C2 = 199384 Sat Oct 11 17:01:39 CDT 2003

P1_F2 Sat Oct 11 17:03:49 CDT 2003

C3=143 Sat Oct 11 17:03:49 CDT 2003

P1_F3 Sat Oct 11 17:04:13 CDT 2003

C4=5 Sat Oct 11 17:04:13 CDT 2003

P1_F4 Sat Oct 11 17:04:38 CDT 2003

C5=0 Sat Oct 11 17:04:39 CDT 2003

P1_F5 Sat Oct 11 17:04:57 CDT 2003

P1_F6 Sat Oct 11 17:04:57 CDT 2003

P1_F7 Sat Oct 11 17:04:57 CDT 2003

P1_F8 Sat Oct 11 17:04:57 CDT 2003

Time for partition 1 = 199966 Sat Oct 11 17:04:57 CDT 2003

C2 = 197612 Sat Oct 11 17:04:59 CDT 2003

P2_F2 Sat Oct 11 17:07:12 CDT 2003

C3=156 Sat Oct 11 17:07:12 CDT 2003

P2_F3 Sat Oct 11 17:07:36 CDT 2003

C4=10 Sat Oct 11 17:07:37 CDT 2003

P2_F4 Sat Oct 11 17:08:01 CDT 2003

C5=0 Sat Oct 11 17:08:02 CDT 2003

P2_F5 Sat Oct 11 17:08:20 CDT 2003

P2_F6 Sat Oct 11 17:08:20 CDT 2003

P2_F7 Sat Oct 11 17:08:20 CDT 2003

P2_F8 Sat Oct 11 17:08:20 CDT 2003

Time for partition 2 = 202888 Sat Oct 11 17:08:20 CDT 2003

Time for global = 2359 Sat Oct 11 17:08:22 CDT 2003

BP=752 Sat Oct 11 17:08:24 CDT 2003

AP=278 Sat Oct 11 17:08:24 CDT 2003

F2 Sat Oct 11 17:09:10 CDT 2003

P3 Sat Oct 11 17:09:59 CDT 2003

F3 Sat Oct 11 17:09:59 CDT 2003

P4 Sat Oct 11 17:10:06 CDT 2003

F4 Sat Oct 11 17:10:06 CDT 2003

P5 Sat Oct 11 17:10:13 CDT 2003

F5 Sat Oct 11 17:10:13 CDT 2003

Time for frequent = 110998 Sat Oct 11 17:10:13 CDT 2003

Complete Sat Oct 11 17:10:13 CDT 2003

Subsets Sat Oct 11 17:10:13 CDT 2003

Rules Sat Oct 11 17:10:14 CDT 2003

Time for rules= 1625 Sat Oct 11 17:10:15 CDT 2003

The second column, in each row is the timestamp when all the tuples where inserted in that particular relation. The first column contains the relation name and their cardinality. For those relation names, which do not have "=" character in them, they are either the relations for Frequent itemsets (F_k) or were not generated but are there as the variable Stop Level, in the configuration file, specifies that the experiment should run until that pass number. (We do so just to maintain consistency in the output that is generated). The file also contains the time it takes during each phase of the execution. For example, for the partitioned approach it has the time it takes to generate the frequent itemsets for each partition, the time to merge and the time taken to generate the global frequent itemsets. The cardinalities of frequent itemsets relations are calculated at the end during writing the ResultLog.) The other log (ResultLog) is written only when a given approach completes successfully. It contains the number of Frequent itemsets (F_k) generated for each data set.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this thesis, we have focused on the partitioned and incremental approach to association rule mining useful for multiple databases. We have presented a partitioned approach to association rule mining, which is appropriate to mine data stored in multiple DBMSs. The incremental approach proposed in this thesis reduces the task of recomputing the rules when the data in the DBMS changes.

The partition algorithm described provides an efficient way of discovering association rules in large database. It is convenient to use this algorithm when there are multiple databases because the amount that has to be transferred by using this approach is comparatively less than transferring all the raw data to a single DBMS for performing mining. The TIDLIST approach, which was used in the main-memory based partition algorithm, had drawbacks when used in the database approach. This thesis presented two extensions -- Approach I and Approach II using the negative border concept, which are suitable for multiple databases.

The incremental association rule mining algorithm aids in the generations of the frequent itemsets when the data in the DBMS changes. The negative border concept used by the algorithm helps in determining the frequent itemsets when the support and confidence

changes. When the negative border expands a pass over the database has to be made to generate the support count. This turns out to be time consuming in the database approach. So we have proposed to update the negative border incrementally which can be used for support counting in the case where the negative border expands. Extensive experiments have been performed for the partitioned and incremental approached on Oracle 8i and IBM DB2.

REFERENCES

1. Zhang, S., X. Wu, and C. Zhang, *Multi-Database Mining.* IEEE Computational Intelligence Bulletin, Vol. 2, No. 1, June 2003: p. 5-13.
2. Wu, X. and S. Zhang. *Synthesizing High-Frequency Rules from Different Data Sources*. in *IEEE Transactions on Knowledge and Data Engineering*. 2003.
3. Han, J. and M. Kamber, *Data Mining : Concepts and Techniques*. 2001: Morgan Kaufmann Publishers.
4. Thomas, S., et al. *An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases*. in *Knowledge Discovery and Data Mining*. 1997.
5. Thomas, S., *Architectures and optimizations for integrating Data Mining algorithms with Database Systems*, in *CSE*. 1998, University of Florida: Gainesville.
6. Thomas, S. and S. Chakravarthy. *Incremental Mining of Constrained Associations*. in *In Proc. of the 7th Intl. Conf. of High Performance Computing (HiPC).* 2000.
7. Thuraisingham, B., *A Primer for Understanding and Applying Data Mining.* IEEE, 2000. Vol. 2, No.1: p. 28-31.
8. Agrawal, R., T. Imielinski, and A. Swami. *Mining Association Rules between sets of items in large databases*. in *ACM SIGMOD International Conference on the Management of Data.* 1993. Washington, D.C.
9. Agrawal, R. and R. Srikant. *Fast Algorithms for mining association rules.* in *20th Int'l Conference on Very Large Databases (VLDB).* 1994.
10. Savasere, A., E. Omiecinsky, and S. Navathe. *An efficient algorithm for mining association rules in large databases*. in *21st Int'l Cong. on Very Large Databases (VLDB).* 1995. Zurich, Switzerland.

11. Chen, Y., *An Efficient Parallel Algorithm for Mining Association Rules in Large Databases*. 1998, Georgia Institute of Technology: Atlanta.

12. Shenoy, P., et al. *Turbo-charging Vertical Mining of Large Databases*. in *ACM SIGMOD Int'l Conference on Management of Data.* 2000. Dallas.

13. Han, J., J. Pei, and Y. Yin. *Mining Frequent Patterns without Candidate Generation*. in *ACM SIGMOD Int'l Conference on Management of Data.* 2000. Dallas.

14. Houtsma, M. and A. Swami. *Set-Oriented Mining for Association Rules in Relational Databases*. in *11th International Conference on Data Engineering (ICDE).* 1995.

15. Han, J., et al. *DMQL: A data mining query language for relational database*. in *ACM SIGMOD workshop on research issues on data mining and knowledge discovery*. 1996. Montreal.

16. Meo, R., G. Psaila, and S. Ceri. *A New SQL-like Operator for Mining Association Rules.* in *Proceedings of the 22nd VLDB Conference*. 1996. Mumbai, India.

17. Agrawal, R. and K. Shim, *Developing tightly-coupled Data Mining Applications on a Relational Database System*. 1995, IBM Almaden Research Center: San Jose, California.

18. Sarawagi, S., S. Thomas, and R. Agrawal. *Integrating Association Rule Mining with Relational Database System: Alternatives and Implications*. in *ACM SIGMOD Int'l Conference on Management of Data.* 1998. Seattle, Washington.

19. Hongen, Z., *Mining and Visualization of Association Rules over Relational DBMSs*, in *CSE*. 2000, UFL: Gainesville.

20. Dudgikar, M., *A Layered Optimizer or Mining Association Rules over RDBMS*, in *CSE Department*. 2000, University of Florida: Gainesville.

21. Mishra, P. and S. Chakravarthy. *Performance Evaluation and Analysis of SQL-92 Approaches for Association Rule Mining*. in *BNCOD Proc.* 2003.

22. Oracle, *Java Stored Procedures Developers Guide*., http://otn.oracle.com/doc/oracle8i_816/java.816/a81353/toc.htm.

23. Oracle, *JDBC Developers Guide*., http://otn.oracle.com/docs/products/ias/doc_library/1022doc_otn/apps.102/a83724.pdf.

24. Toivonen, H. *Sampling Large Databases for Association Rules.* in *In Proc. 1996 Int. Conf. Very Large Data Bases*. 1996: Morgan Kaufman.

25. Mishra, P. and S. Chakravarthy, *Evaluation of K-way Join and its variants for Association Rule Mining*. 2002, Information and Technology Lab at The University of Texas at Arlington, TX.

BIOGRAPHICAL INFORMATION

Hima Valli Kona was born on December 16, 1978 in Vijayawada, India. She received her Bachelor of Engineering degree in Computer Science and Engineering from Madurai Kamaraj University, Tamil Nadu, India in May 2001. In the Fall of 2001, she started her graduate studies in Computer Science and Engineering at The University of Texas, Arlington. She received her Master of Science in Computer Science and Engineering from The University of Texas at Arlington, in December 2003. Her research interests include Data Mining and Web technologies.