Towards Ad Hoc Mining of Association Rules with Database Management Systems

Tok Wee Hyong, Indriyati A. and Low Wai Lup
School of Computing
National University of Singapore
{tokweehy, indriyat, lowwailu}@comp.nus.edu.sg

October 20, 2000

Abstract

Current data mining techniques present impenetrable black-box interfaces to users with little chances of interaction during the mining process. We advocate the need for *ad hoc* mining facilities which allow users to get a feel of the final data mining results quickly without having to materialize input data and going through the whole mining process. We investigate the feasibility of ad hoc mining of association rules. The mining can be performed on views defined over relations in database management systems to investigate "what-if" scenarios. The results can help users decide whether to proceed with the mining project. We present a prototype which implements the popular Apriori algorithm as a mining operator within the Predator database management system. A variant of DMQL and a cost-based mining query optimizer are designed to incorporate the use of this new operator into Predator. Experiments show that this approach is indeed viable for ad hoc data mining with response times adequate for interactive data mining.

Keywords : Mining association rules, Database management systems, Interactive data mining

1 Introduction

The amount of data being stored in databases and managed by database management systems (DBMS) is increasing at an explosive rate, mirroring the growth of the Internet. There is now a huge demand for the nuggets of information stored in these datawarehouses (or datamarts) that can be discovered through data mining. However, data mining

techniques were developed traditionally for small datasets stored in flat files. These techniques cannot leverage on the years of work by the database research community in the areas of query optimization, concurrency control, data pipe-lining among others.

In recent years, many researchers have started to focus on issues arising from integrating mining operations into databases. These include developing query languages to incorporate data mining functions [JHZ96] [TIA96], architectures and the performance of different levels of coupling [SS98].

Most of existing data mining systems that are integrated with database management systems (DBMS) follow the loose-coupling approach. Often, the DBMS is simply used as a data store from which data is fetched through the use of cursors exposed through a interface such as Open Database Connectivity/Java Database Connectivity (ODBC/JDBC), and the tuples returned are stored in more efficient data structures outside the DBMS.

In this work, we investigate the viability of incorporating data mining operators at the same level as other data operators, such as the unbiquitous *order-by*, *group-by*, and *join* operators. This allows dynamic view definitions over several relations by users to investigate 'what-if' scenarios and obtain data mining results quickly. Sample results can be obtained from a sub-set of the data without having to go through the expensive and time consuming data preparation stage. Our implementation extends Predator's [Ses97] [DoCS00] query language with a variant of the DMQL [JHZ96] for expressing mining queries.

The rest of this paper is organized as follows. Some concepts necessary for the understanding of later sections are explained in Section 2. The motivation and rationale of ad hoc mining is discussed Section 3. We present a survey of existing data mining and DBMS integration frameworks and systems in Section 4. In Section 5, we discuss the architectural design and implementation of a association rule mining operator for Predator, extending Predator's query language with a variant DMQL and a cost-based approach for optimization. We explain our experiments and results in Section 6. We conclude in Section 7.

2 Background

We cover briefly the concepts of Association Rule Mining and the popular Apriori Algorithm. Interested readers can refer to [AS94] for full details.

2.1 Association Rule Mining

Given a transaction, where each transaction is a set of items, an association rule is an expression $X \rightarrow Y$, where X and Y are items. The meaning of such a rule is that the transactions that contain the items in X tend to also contain the item in Y. The problem of mining association rules is to find all rules that satisfy a user-specified minimum support and minimum confidence.

Thus, the association rule mining is decomposed intto two subproblems:

- Find all combination of frequent itemsets whose support is greater than minimum support
- Use the frequent itemsets to generate the rules.

2.2 Apriori algorithm

The Apriori algorithm is used to find all frequent itemset refering to the first subproblem. It makes multiple passes over the database. In the first pass, the algorithm simply counts the item occurences to determine the frequent 1-itemsets. A subsequent pass, k, consists of two phases. In the first phase, the frequent itemsets L_{k-1} found in the $(k-1)$th pass are used to generate the candidate itemsets C_k using the *apriori_gen()* function. This first function joins L_{k-1} with L_{k-1}, the join condition being that the lexicographically ordered first $k-2$ items are the same. Next, it deletes all those itemsets from the join result who have some $(k-1)$-subset that is not in L_{k-1}, yielding C_k. In the second phase, the algorithm scans the database. For each transaction, it determines which of the candidates in C_k are contained in the transaction using a hash-ree data structure and increments their count. At the end of the pass, C_k is examined to determine which of the candidates are frequent, yielding L_K. The algorithm terminates when L_K becomes empty.

3 Motivation

We define "*ad hoc data mining*" to be a flexible and interactive data mining process performed over a sub-set of the data without the need to perform data preparation with the aim of getting a 'feel' of the results that can be achieved with the whole-scale mining process.

Data mining is an iterative process, that demands large investments of time and other resources. It is well acknowledged that most of the time in Knowledge Discovery and Data Mining (KDD) processes is spent on data preparation which includes data integration, data cleaning and conflict resolution. All these resources are required even before the start of the mining process and might be wasted if the mining project if changed or terminated later. Ad hoc mining allows users to get sample mining results without sinking in all these resources.

'What-if' scenarios can be investigated easily simply through the interactive nature of ad hoc mining. Questions like "Should we add in the quantity for each item in the transactions? And how will this affect the rules generated? How will the results be affected if I change the threshold values?" can be answered quickly using data subsets to aid decision making for attributes to be used in the real process.

As most data of organizations are stored in some form of DBMS, it makes sense to perform such preliminary analysis directly on existing relations. Ad hoc mining of small subsets of data can even be performed on views defined over multiple relations over operational tables.

One attraction of SQL implementation is inter-operability and usage flexibility [SS98]. The ad hoc querying support provided by a DBMS enables flexible usage and exposes potential for pipelining the input and output operators of the mining process with other operators in the DBMS. However, to exploit this feature one needs to implement the mining operators inside the DBMS. This will require major reworking of existing database systems.

4 Related Works

In [SS98], performance of various techniques on coupling data mining operations with database systems were studied. Amongst the techniques studied include: (1) Association rule mining operations expressed as a series of SQL statements. (2) Use of UDFs (3) Use of Stored Procedures (4) Cache-Mine in which tuples read from a database is stored in efficient data structures (5) SQL-OR functionalities provided by IBM DB2 (i.e GatherJoin, GatherCount, GatherPrune). It is interesting to note that by expressing the mining operations in SQL, it leads to greater usage flexibility and inter-operability. However, since the mining operations is expressed as a series of SQL statements, it lowers the chance of pipelining the input and output operators of the mining process with other operators in the DBMS. In order to fully exploit pipelining amongst operators, we would have to implement the mining operations as an operator in the DBMS based on the iterator interface proposed by [Gra93].

A data mining architecture which is tightly coupled with the RDBMS, is studied in [SN99]. A framework, based on query flocks was presented. Esentially, a query flock is a generate-and-test system, in which a family of queries that are identical except for the values of one or more parameters are asked simultaneously. It consists of a query flock compiler that will transform a complex data mining query into a query flock plans which consists of sequences of simple SQL queries. In the query flock technique proposed in [SN99], an external optimizer complements the existing query optimizers that make optimization decisions more effectively because it takes the task of data mining into consideration. Except for a new logical operator group-select, no new physical operators were introduced into the physical execution plan. This is in contrast to our work on adding of new mining operators into the physical execution plan to support data mining.

In [RA95] and [RA96], a methodology for tightly coupling applications to the IBM DB2/CS relational database system was presented. The technique proposed selectively push parts of a applications that access data records and perform computations into the database through the use of user-defined functions (UDFs). This method is not able to leverage on optimization and pipelining facilities provided by the DBMS and incurs expensive context switching costs through the use of UDFs.

Finally, we note that there has been efforts recently to establish industry standards for data mining so that different data mining algorithms from various data mining solution providers can be easily plugged into user applications [Cor00].

5 Extending Predator with Mining Operators

Predator [Ses97] [DoCS00] is an extensible object-relational DBMS, built on top of the Shore storage manager [CDF+94]. In about 65,000 lines of C++ code, Predator is a lightweight, and highly functional OR-DBMS. In recent years, various researchers have made use of Predator in various research projects. These include the implementation of new join operators (Example - threaded symmetric hash join operator which exploits a double pipeline) and the studies of client-side user-defined functions (UDFs). Besides supporting primitive datatypes such as integer, double and string, Predator also supports Enhanced Abstract Datatypes (E-ADTs) such as JPEG images, polygons, circles, etc.

In our work on demonstrating the feasibility of ad hoc mining via tight coupling of mining operations within a DBMS, Predator was chosen due to its highly extensible architecture and well-defined codebase. We focus on an end-to-end solution, which consists of:

- Extending the Predator's grammar to support mining queries using Flex/Bison [DS95] [Pax95](based on a DMQL-variant).
- Defining a new mining query optimizer (A cost-based approach is used when considering access path for the base relations)
- Implementing a new iterator-based mining operator that performs association rule mining

5.1 Extending the Query Language - DMQL variant

In the current literature, there exists various language proposals to extend SQL to support mining operators. The M-SQL language [TIA96] extends SQL with a unified operator, *Mine*, to generate and query a whole set of propositional rules. DMQL [JHZ96] extends SQL with the syntax with support mining of characteristics rules, discriminant rules,

```
find association rules
related to item
with key tid
from trans
where price>15 and price<205
with support threshold = 0.05
with confidence threshold = 0.7
```

(a)

```
find association rules
related to item
with key tid
from (
   select * from trans
   where price>15 and price<205 )
      trans_new
with support threshold = 0.05
with confidence threshold = 0.7
```

(b)

Figure 1: A sample query.

classification rules, and associations rules. In addition, OLE DB for DM (data mining) [Cor00] applications was proposed recently by Microsoft to provide an industry standard for data mining so that different data mining algorithms from various data mining tools can be easily plugged into user applications. OLE DB for DM introduces one new virtual object, referred to as the data mining model (DMM), as well as several new commands for manipulating the DMM. Of all these proposed language extensions, we choose to model the query language based on DMQL for its simplicity, and ease in integrating with Predator's existing SQL grammer.

In addition, we note that in order to support a variety of ad-hoc mining queries, nested queries must be supported by the mining language. Nested queries allow the mining of the results from other relational operations, such as a join. Hence, the *from* clause must allow the composition of an inner SQL query.

Our design of the data mining language used in Predator is based on DMQL [JHZ96]. Figure 1a shows how a association rule mining query can be expressed in the DMQL-variant. In the query, we wish to find association rules from the table *trans* such that the price of the item is between 15 and 205 (exclusive), with a support of 0.05 and a confidence of 0.7. Figure 1b shows how the same query can be expressed as a nested query supported by our implementation.

The enrichment of Predator's SQL grammar is achieved by modifying the existing SQL grammar file and generating the new SQL grammar using Flex/Bison. Figure 2 presents

```
mine_block:= find_clause
             related_to_clause
             withkey_clause
             from_clause
             where_clause
             support_threshold_clause
             confidence_threshold_clause

withkey_clause:= XXX_TOK_WITH XXX_TOK_KEY projection_list
find_clause:= XXX_TOK_FIND XXX_TOK_STRING XXX_TOK_RULE
related_to_clause:= XXX_TOK_RELATED XXX_TOK_TO projection_list
support_threshold_clause:= XXX_TOK_WITH XXX_TOK_SUPPORT
                           XXX_TOK_THRESHOLD
                           XXX_TOK_EQ XXX_TOK_DOUBLE
confidence_threshold_clause:= XXX_TOK_WITH XXX_TOK_CONFIDENCE
                              XXX_TOK_THRESHOLD
                              XXX_TOK_EQ XXX_TOK_DOUBLE
```

Figure 2: Grammar for DMQL-variant expressed in BISON's syntax

the DMQL-variant grammar implemented. The identifiers prefixed by XXX_TOK_ are tokens returned by the scanner which is defined using FLEX (similar to LEX). Projection_list, from_clause, where_clause are defined by the existing Predator's grammar [Ses97] (and hence have been omitted).

5.2 Mining Query Optimizer

The goal of query optimization is to find a good evaluation plan for a user query. Many techniques have been proposed towards query optimization in DBMSes. However, as most of the data mining systems discussed in literature are loosely coupled, there is little work on query optimization for mining queries in relational databases. In retrospect, the role of mining query optimization cannot be neglected if mining operators are to be tightly integrated with a DBMS. For example, [RR00] discussed the idea of iceberg queries, and showed how traditional query optimizers failed to produce a good evaluation plan for the query. [SN99] proposed the use of an external optimizer that sits on top of the DBMS, which breaks a complex data mining query into a sequence of smaller queries that can be executed efficiently at the database. In addition, a mining query optimizer can generate

plans based on statistics (e.g histogram-based) or the nature of the mining query. A particular mining algorithm could be identified and used during the actual execution.

Query optimization in Predator proceeds in the following manner. First, a SQL query is broken into a series of optimization blocks (e.g. Select-Project-Join(SPJ), GroupBy, OrderBy blocks). Query optimization is performed intra-block. Various optimizers (i.e Naive, System-R style, KBZ) are defined within Predator and this allows researchers to specify the optimizer to be used in order to study the performance of various optimization algorithms. Optimization across blocks (i.e inter-block optimizations) is entirely based on heuristics (making use of interesting orders). Query execution in Predator proceeds in a pipelined manner. Physical operators defined in Predator follow the iterator interface proposed in [Gra93]. Output from an operator is pipelined (as input) to another operator.

Notably, the primary focus of query optimization in Predator is on the select-project-join (SPJ) query blocks. Predator provides many in-built optimizers with different degress of complexity. (i.e Naive, Semi-Naive, Dynamic - System-R and Greedy). Whenever a query is executed, the parsed query plan is then passed to one of these optimizers depending on the type of query optimizer specified. The flexibility of being able to specify which optimizer to be used allows a researcher to test new query optimization techniques.

In most loosely coupled data mining systems, a full filescan is often initiated in order to perform task such as association rule mining, ignoring available indices. The data retrieved from the filescan is then stored in a more efficient data structure to faciliate the mining operations. It can be argued that in order to reap the full benefits of tight integration of mining operations within a DBMS, we should not only look at developing operators for each mining tasks that fits into the iterator paradigm, but also consider the issues of optimiziation of the mining query for efficient operation. These optimization decisions can be based on if it is advantageous to make use of available indexes built on the underlying data (in the case of a base relation is being mined), or passing the task of query optimization to a SPJ optimizer (in the case of nested queries).

In our implementation, we provide a simple cost-based query optimizer which generates efficient access plans for the underlying base relation to be mined. A new mining query optimizer (i.e RelMineOptimizer) is defined which optimizes the access methods

```
Optimization steps:
S = {}
If ( Base Relation ) {
S = S U {Full-File-Scan}
For each predicate Pi {
  If there is an index built on Pi
   S = S U {Index-ScanPi}
 }
 p = Plan in S with minimum cost
}
Else if ( Derived Relation ) {
  Invoke SPJ query optimizer for
  SPJ block

  p = Plan obtained from SPJ
  query optimizer
}
return p
```

Figure 3: Pseudocode for Mining Query Optimization

for the base relation referenced in the *from* clause. As nested queries (i.e the *from* clause could potentially be another SQL query) are supported, we handle the optimization for the inner query using the existing query optimizers in Predator. It is important to note that query optimization in Predator is performed within a block (i.e Mine, SPJ, GroupBy, Orderby block), and optimization across blocks is based on heuristics. The pseudocode for the optimizer is shown in Figure 3.

Lastly, if selection predicates are specified in the where clause of the query, these are pushed to the relation scan. In comparison to a loosely-coupled approach where each record (read from a file), has to be scanned before deciding whether the record should be discarded, pushing selection allow us to reduce the data prior to the data mining task.

5.3 Mining Operator

Once a plan is produced by the query optimizer, Predator proceeds to generate the respective operators (e.g SortMerge operator, file scan operator) for the plan. This proceeds in a top-down manner, in which the GeneratePlanOp() method of the top operator is invoked, which creates the run-time component for the top operator. The top operator

then proceeds to invoke the GeneratePlanOp() method of its child operators, and so on.

All operators in Predator follow the iterator-interface, which implements 3 common methods. These methods are: *InitHandle* (i.e setting up the data structures required for run-time), *GetNextRecord* (i.e producing the next record), and *CloseHandle* (i.e reclaiming the resources used during run-time). During query execution, the *InitHandle* method of the top operator is invoked, and this propagates down to its child operator. Similarly, to produce a record, the *GetNextRecord* of the top operator will be invoked and proceeds to obtain the next record from its child operators.

In order to support association rule mining in Predator, the RelMinePlanOp operator is implemented to perform association rule mining. The implementation of the RelMinePlanOp is based on the program in [CHR00], which provides the basis for a fast implementation of the Apriori algorithm based on the use of a prefix tree. The original C code was wrapped within the RelMinePlanOp class, and the input streams for the Apriori algorithm was modified to read from a character stream, instead of a file. The reasons for providing a wrapper class are two-fold: First, we wish to demonstrate that an existing mining algorithm can be easily wrapped and integrated as a relational operator within Predator. Second, we wish to perform a fair comparison of a tightly-coupled approach (mining operator) versus a loosely-coupled approach (having the data source as a file) by basing it on the same algorithm. This minimizes bias that can result due to differences in implementation of the algorithm. In a later section, we will present empirical results based on three different approaches. Figure 4 shows how the new mining operator fit into Predator with its existing operators.

6 Experimental results

To assess the effectiveness of our approach, we empirically compared the performance of our tight coupling approach with two other approaches. The 3 approaches used in the experiments are : (1) Tight coupling through integrating a new operator to support association rule mining within an existing relational DBMS, Predator. (2)Running Apriori algorithm on a delimited text file residing on a file system. (3) Simulating a loose-coupling approach using Oracle and an external association rule mining program.

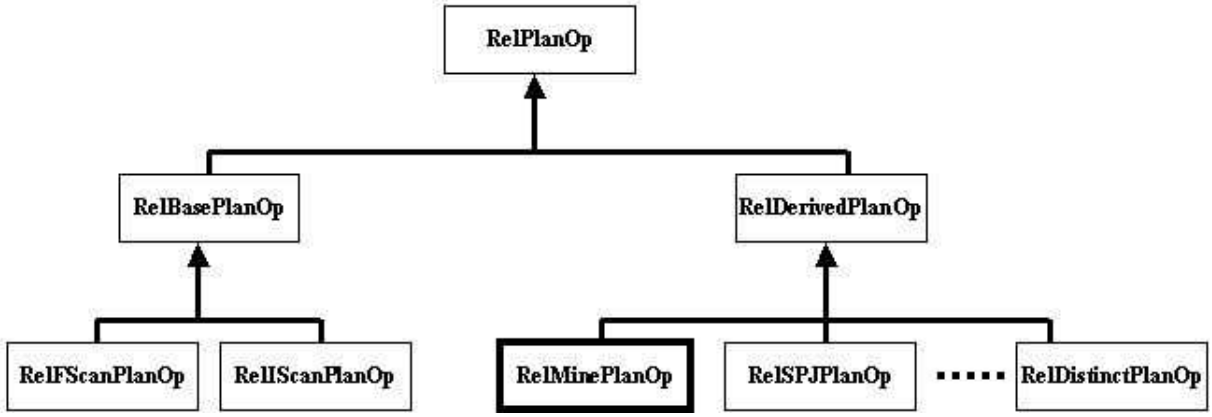

Figure 4: New mining operator in Predator

Table 1: Charateristics of the Datasets

Name	# of tuples	# of transactions
Data Set 1 - Car	12096	1728
Data Set 2 - Iris	750	150
Data Set 3 - Segmentation	4200	210
Data Set 4 - Wine	2792	178
Data Set 5 - Zoo	1818	161
Data Set 6 - Breast Cancer	18208	569

For this loose-coupling approach, we created an JAVA application which obtains the data from the Oracle server via JDBC. After the data arrives at the client application, the Apriori algorithm is invoked as a separate process. The machine used for all the experiments is a Intel Pentium III PC with a CPU clock speed of 667 MHz, and 128 MB of main memory. We use the same implementation of the Apriori algorithm for all three approaches to eliminate any performance differences due to differences in implementation of the algorithm.

Six real-life datasets were used in the experiments. These datasets were obtained from UCI Machine Learning Repository. Table 1 summarizes the charateristics of these datasets.

For each data set, we measured the execution times of all the three approaches for different combinations of support and confidence levels.

Figure 5 summarizes the experimental results. The x-axis shows the four different combinations of support and confidence level for each experiment. The y-axis is the execution time of each approach measured in seconds. The different combinations of

minimum support and confidence were chosen to allow a reasonable number of passes and frequent itemsets.

We see that in all cases, the file system Apriori approach performs the best and the loose-coupling approach is the worst. For the loose-coupling approach, the client application needs to wait until the server has finished transferring the data. This data shipping accounts for the largest percentage of the execution time.

Our approach performs better than the loose-coupling approach. However, it still is slower than the file system approach. This is due to several reasons. As we extended Predator's cost-based query optimizers with a new mining optimizer, the optimizer (refer to Section 5.2) will have to decide on a good access and execution plan (e.g. weighing the advantages and costs of the use of indices and possibility of pushing selection predicates). This takes time, even for simple queries, although its benefits might not be obvious until the queries get more complicated. In addition, due to the client-server architecture of Predator, there will be costs involved in setting up communication channels and the actual communication between the various components (storage manager, operators and etc). Due to all these factors, the performance of our approach is slightly slower than running the Apriori algorithm on the file system. However, the benefits of riding the data mining operations on top of a DBMS (refer to Section 3) heavily outweighs the additional costs involved. Our results agree with the observations made in [SS98] where they noted that running against flat files is typically a factor of five to ten faster compared to running against data stored in DBMS tables.

The discrepancy in time and resources taken up by the above-mentioned processes can be minimized through the careful use of selection predicates to minimize the size of the sample data set used for the ad-hoc mining activity. The availability of strategically built indices to speed up reading of data by the DBMS will also help. The flat file implementation of Apriori actually offers the upper performance bound in this aspect since the data from the file is likely to exist as contiguous blocks on the disk.

Figure 6 illustrates the effect of increasing the number of records on the execution speed. We observe that the performance of the file system Apriori is almost unaffected by the number of records existing in the transaction. Both our approach and the loose-coupling implementation shows an linear increase in the time needed to perform associa-

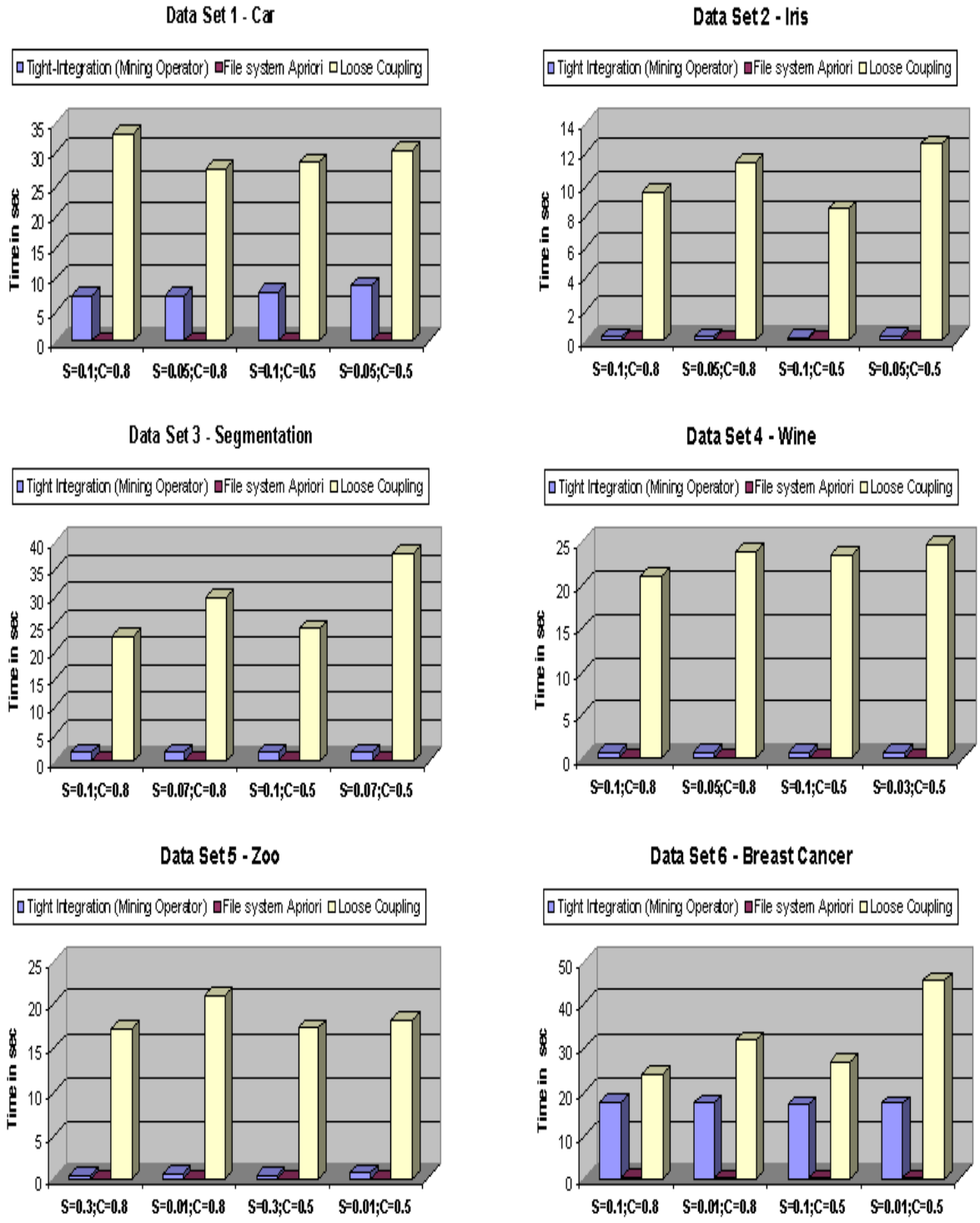

Figure 5: Comparison of the three approaches: Tight coupling via integration of new mining operator in Predator, Apriori on file system and Loose Coupling using JDBC on Oracle, on six real-life dataset. For each dataset, four different support and confidence values are used.

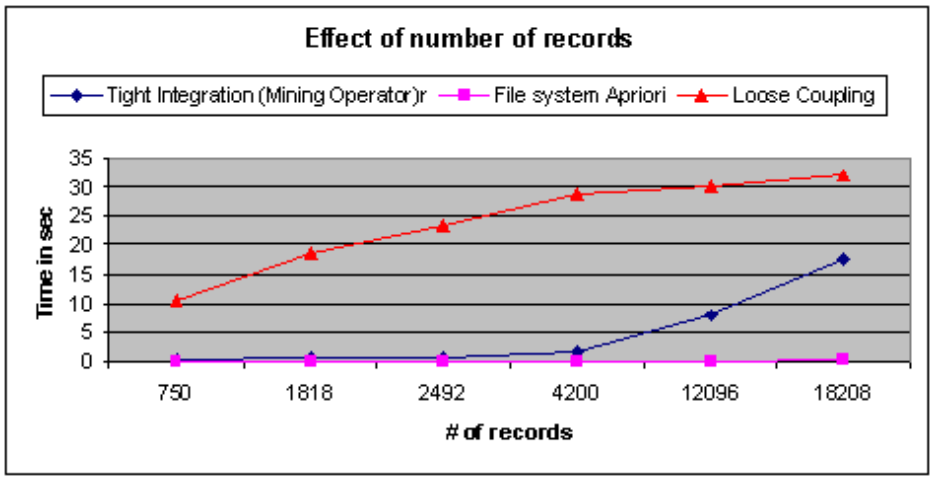

Figure 6: Effect of number of records on execution time

tion rule mining. We also noted that our approach performs better than the loose-coupling approach consistently.

7 Conclusion

We have presented the arguments for ad hoc mining and implemented a prototype which defines a new association rule mining operator within an existing DBMS. Our contributions include a three-step approach for integrating a new mining operator with an existing DBMS: (1) extending the query language, (2) defining a new query optimizer and (3) implementing the mining operator. Our experiments show the performance of this approach is better than existing loose-coupling methods, with results being returned in real/near real time for small/medium-sized datasets.

Future work in this area of research includes integrating other data mining activities (clustering, feature selection etc.) with various DBMSes. There has been work on extending the Apriori algorithm with multiple minimum supports. It is interesting to investigate if mining operators can easily accomodate such extensions.

References

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and

Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.

[CDF+94] M. J. Carey, D. J. Dewitt, M. J. Franklin, N. E. Hall, M. L. McAuliffe, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. K. Tan, O. G. Tsatalos, S. J. White, and M. J. Zwilling. Shoring up persistent applications. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):383–394, June 1994.

[CHR00] Apriori implementation based on prefix tree, 2000. Available at URL `[http://fuzzy.cs.Uni-Magdeburg.de/ borgelt/]`.

[Cor00] Microsoft Corporation. Ole db for data mining specification version 1.0, July 2000.

[DoCS00] Cornell University Dept. of Comp. Sci. Predator, 2000. Available at URL `[http://www.cs.cornell.edu/predator/]`.

[DS95] Charles Donnelly and Richard Stallman. Bison, the yacc-compatible parser generator, 1995. Available at URL `[http://www.gnu.org/manual/bison/]`.

[Gra93] Goetz Grafe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 15(2):73–169, June 1993.

[JHZ96] Wei Wang Krzysztof Koperski Jiawei Han, Yongjian Fu and Osmar Zaiane. Dmql: A data mining query language for relation databases. *Proceedings of 1996 SIGMOD workshop on research issues on data mining and knowledge discovery, Montreal, Canada*, May 1996.

[Pax95] Vern Paxson. Flex - a fast scanner generator, 1995. Available at URL `[http://www.gnu.org/manual/flex/]`.

[RA95] K. Shim R. Agrawali. Developing tightly-coupled applications on ibm db2/cs relational database system: Methodology and experience. 1995.

[RA96] K. Shim R. Agrawali. Developing tightly-coupled data mining applications on a relational database system. *Proceedings of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining, Portland, Oregon*, August 1996.

[RR00] Johannes Gehrke Raghu Ramahkrishnan. Database management systems, 2nd edition, 2000.

[Ses97] Praveen Seshadri. Predator - design and implementation. November 1997.

[SN99] Dick Tsur Svetlozar Nestorov. Integrating data mining with relational dbms: A tightly-coupled approach. *Proceedings of 4th Workshop on Next Generation Information Technologies and Systems, NGITS '99*, July 1999.

[SS98] Rakesh Agrawal Sunita Sarawagi, Shiby Thomas. Integrating association rule mining with relational database systems: Alternatives and implications. *Proceedings of SIGMOD '98*, 1998.

[TIA96] A. Virmani T. Imielinski and A. Abdulhani. Discovery board application programming interface and query lanaguage for database mining. *Proceedings of*

the 2nd Int'l Conference on Knowledge Discovery and Data Mining, Portland, Orregon, August 1996.