DRILA: A Distributed Relational Inductive Learning Algorithm

SALEH M. ABU-SOUD
Computer Science Department
New York Institute of Technology
Amman Campus
P.O. Box (1202), Amman, 11941
JORDAN
sabusoud@nyit.edu

ALI AL-IBRAHIM
Faculty of Information Systems
The Arab Academy for Banking and
Financial Sciences
P.O. Box. 1470, Amman, 11118
JORDAN
alikitim@yahoo.com

Abstract:- This paper describes a new rule discovery algorithm called Distributed Relational Inductive Learning DRILA, which has been developed as part of ongoing research of the Inductive Learning Algorithm (ILA) [11], and its extension ILA2 [12] which were built to learn from a single table, and the Relational Inductive Learning Algorithm (RILA) [13], [14] which was developed to learn from a group of interrelated tables, i.e. a centralized database. DRILA allows discovery of distributed relational rules using data from distributed relational databases. It consists of a collection of sites, each of which maintains a local database system, or a collection of multiple, logically interrelated databases distributed over a computer network. The basic assumption of the algorithm is that objects to be analyzed are stored in a set of tables that are distributed over many locations. Distributed relational rules discovered would either be used in predicting an unknown object attribute value, or they can be used to extract the hidden relationship between the objects' attribute values. The rule discovery algorithm, developed, was designed to use data available from many locations (sites), any possible 'connected' schema at each location where tables concerned are connected by foreign keys. In order to have a reasonable performance, the 'hypotheses search' algorithm was implemented to allow construction of new hypotheses by refining previously constructed hypotheses, thereby avoiding the work of re-computing.

Unlike many other relational learning algorithms, the DRILA algorithm does not need its own copy of distributed relational data to process it. This is important in terms of the scalability and usability of the distributed relational data mining solution that has been developed. The architecture proposed can be used as a framework to upgrade other propositional learning algorithms to relational learning.

Key-words: Distributed Relational Rule Induction, Rule Selection Strategies, Inductive Learning, ILA, ILA2, RILA, DRILA.

1 Introduction

Most computer algorithms have been designed for situations in which all relevant data are stored at a single computer site. This is the classical model of a computer based information and control system. The emerging networked knowledge environment requires a significant move away from this classical model. In these situations of geographically distributed but networked systems, the data relevant for a computation may exist in a number of different databases residing at different network sites. An efficient system for computations with such distributed data would work by doing as much work at local sites as possible and then communicating minimum required information among the sites. This is much more efficient than transferring the complete databases to a single site, join these databases, and then execute algorithms with this data. They require each object to be described by a fixed set of attributes. Compared to a single table of data, a distributed relational database containing multiple tables that are distributed over network to several locations makes it possible to represent more complex and structured data. For these reasons, it is important to have discovery algorithms running for distributed relational data in its natural form without requiring the data to be viewed in a single table at the same location. A distributed relational data model consisting of multiple tables at each location over network may represent several object classes, i.e. within a schema while one set of tables represents a class of object, a different set of tables may represent another class. Before starting

discovery processes, users should analyze the schema and select the list of tables that represents the kind of objects they are interested in. One of the selected tables will be central for the objects and each row in the table should correspond to a single object in the database. This central table is named as 'target table' [1] and [2], 'primary table' [3], 'master relation' [4], or 'hub table' [5].

As a matter of fact, an efficient management [6], [7] could get a lot of benefits by using a high technology of distributed relational data, such as indexing, query services and transaction management support, also it can structure more complex data. In contrast to a single table of data, these systems make it possible to represent more complex and structured data. As a result of their advantages over other ways of storing and managing data, a significant amount of current scientific and commercial data is stored in distributed relational databases. Theoretically, any distributed relational database can be transformed into a single universal relation to get the benefit of traditional data mining systems. However, in practice this can lead to relations of unmanageable sizes especially when there are recursive relations in the schema. Because relational data can result in a combinatorial explosion in either the number of instances or the number of attributes [8] depending upon whether one decides to duplicate or aggregate. For this reason, it is important to have learning algorithms running for distributed relational data without requiring the data to be viewed in one single table.

Previous related work on single tables like *ILA* [11] , *ILA2* [12] cannot analyze relational data without first transforming it into a single table, this transformation, however, is not always easy and results in the lost of the structural information that could potentially be useful for the data mining processes or data mining or relational data mining.

This being rightfully highlighted as a field not adequately covered by researchers despite its importance to developing a generalized method by which database problems can be efficiently tackled. Considerable amount of work was done to elucidate the algorithm of *ILA*, by transformation of data into a single table; farther on this *ILA2* was developed to solve the problem of overfitting.

Acknowledging the need to benefit from relational database management systems (RDBMS) in learning algorithms, research was driven once a step forward by implementing a relational database inductive learning algorithm called *RILA* [13] which aims to develop data analysis solutions for relational data without requiring it to be transformed into a single table, but did not put much concentration on solving the learning rules from distributed databases. *RILA* was developed with two rule selection strategies:

1. Select early: inherited from *ILA2* algorithm.
2. Select late: developed with *RILA* so rule selection is performed after the hypothesis search process is completed. It is similar to the rule selection strategies used in well-known relational rule induction algorithms such as the WARMR algorithm [17]. But this effort only considers centralized database systems.

There exists many algorithms in the literature that handle the problem of extracting inductive rules from distributed relational databases from one face either horizontally partitioned datasets as SVM [19], [20], or vertically partitioned datasets [21] . On the other hand, there are few algorithms as the Distributed Decision Tree Algorithm [2], handle the problem from the two faces; horizontally and vertically partitioned datasets, the main problem of this algorithm is that it is a non incremental. This means that if new examples are entered, the decision tree must be built all over again. In addition, there may be more than one decision tree for a given set of examples.

WARMR [17] is an algorithm developed to learn from multiple relations. When Clare [18] wanted to use WARMR to process a relational yeast data set, this was not possible due to the amount of memory required by the system for the data. Due to this limitation of WARMR, a distributed version of the WARMR algorithm, called PolyFARM (Poly-machine First-order Association Rule Miner) was developed to allow processing to be distributed across a cluster of computers [18].

This paper describes an algorithm for learning from distributed relational data stored in, and managed by modern distributed relational database systems. This algorithm is called *Distributed Relational Inductive Learning Algorithm DRILA* that can be used to discover knowledge in the form of relational classification rules. The main contribution here is the adaptation of a traditional propositional learning algorithm to the relational domain and a new effective rule selection strategy. Pruning techniques have also been incorporated into the implementation of the algorithm.

Unlike our approach, traditional relational learning algorithms have been generally designed for relational data stored in Datalog1/Prolog servers. These algorithms are usually called ILP2 based algorithms [9]. Adapting these algorithms for data stored in relational databases is complicated because

Prolog engines are not designed to support relational data stored in distributed relational databases as they support relational data stored in the native Prolog bases. Some algorithms such as the FOIL algorithm [10] have been designed in a generic way, independent of the location of the actual relational data; these algorithms can be adapted for data stored in distributed relational database management systems. However, they generally assume the input data stored in the runtime memory of the learning processes. In order to adapt these algorithms for relational data stored in relational database management systems they should be revised to employ the client-server architecture.

Actually, working on distributed relational database mining is a continuous of an evolving interrelated chain that dates back to the early seminal work of the inductive learning in 1998 [11]. This stage of research has evolved logically from previous substantial accomplishments in the field carried out by numerous research teams. The theme of reasonably coordinated research team work is still retained in the present research exercise that well presumably adds to the work of the teams below:

- *ILA* [11]: inductive learning algorithm for learning data store in single table.
- *ILA2* [12]: fast inductive learning algorithm for learning from single table with solution for overfitting problem (noise-tolerant version of the ILA rule induction algorithm).
- *RILA* [13], [14]: relational learning algorithm from centralized database based on *ILA2* algorithm.

Our general strategy for designing an algorithm for learning from distributed data that is provably exact with respect to its centralized counterpart follows from the observation that most of the learning algorithms use only certain statistics computed from the data D in the process of generating the hypotheses that they output. (A statistic is simply a function of the data; examples of statistics include mean value of an attribute, counts of instances that have specified values for some subset of attributes, the most frequent value of an attribute, etc.) This yields a natural decomposition of a learning algorithm into two components:

1. An information extraction component formulates and sends a statistical query to a data source.
2. A hypothesis generation component uses the resulting statistic to modify a partially constructed hypothesis (and further invokes the information extraction component if needed).

So, *DRILA* has been developed for performing supervised learning by classifying from distributed relational databases, depends on *ILA, ILA2* and *RILA* algorithms which handles both strategies of partitioning the datasets: horizontally and vertically.

DRILA of the system that has been developed was adapted from ILA (Inductive Learning Algorithm) [11]. So, for best understanding of DRILA, one must understand RILA [14] and its descendent algorithms ILA [11] and ILA-2 [12]. ILA is a 'covering' type learning algorithm that takes each class in turn and seeks a way of covering all instances, at the same time excluding instances which are not in the class. There is also an improved version of the ILA algorithm named ILA-2 that uses a penalty factor that helps to produce better results for noisy data [12]. Also there is an adapted version of the ILA-2 algorithm is named Relational-ILA which learns rules from centralized databases.

ILA requires a particular feature of the object under consideration to be used as a dependent attribute for classification. In DRILA, at each location over network, however, the dependent attribute corresponds to the target attribute of the target table. It is assumed that the target table is connected to other tables through foreign key relations. DRILA is composed of initial hypotheses generation, hypotheses evaluation, hypotheses refinement and rule selection steps at each location (site) of the distributed database.

2 The Distributed Relational Inductive Learning Algorithm DRILA

2.1 Definition of Distributed Learning

The problem of learning rules from distributed relational databases with periodical updates can be summarized as follows: Given a data set D, a hypothesis class H and a performance criterion P, the learning algorithm L outputs a hypothesis h € H that optimizes P. In pattern classification applications, H is a classifier, the data D typically consists of a set of training examples. Each training example is an ordered tuple of attribute values, where one of the attributes corresponds to a class label and the remaining attributes represent inputs to the classifier. The goal of learning is to produce a hypothesis that optimizes the performance criterion of minimizing some function of the classification error (on the training data) and the complexity of the hypothesis.

Given the fragments $D_1 \ldots D_n$ of a data set D distributed across the sites 1…n, a set of constraints

Z, a hypothesis class H, and a performance criterion P, the task of the learner Ld is to output a hypothesis h € H that optimizes P using only operations allowed by Z. Clearly, the problem of learning from a centralized data set D is a special case of learning from distributed data where n = 1 and Z = φ.

2.2 An Overview of DRILA Algorithm

Depending on the definition of distributed databases and declarations for that data and its nature and how that data are distributed among locations (horizontal or vertical); we build an inductive learning system for distributed database that is concerned with the following points:

- Data size in every location.
- Distance among locations.
- Nature of data in each location.
- Distributing strategy of data in each location: horizontal or vertical or both.

So upon this information, we may have many strategies for learning from a distributed database. These strategies are discussed as follows, showing the strength aspects and drawbacks of each:

Strategy-1: Merging

Merge all the data sets from all distributed sites in one site then start the learning process. This strategy is not valid for these reasons:

- This strategy takes us back to a single table idea in which ILA learning system can be used, or centralized database idea in which *RILA* learning system can be used.
- Transfer data from all sites to a single site causes the database to loose its structural information and makes it weak.
- Time consuming and less efficient, because it needs transferring the complete databases to a single site, join these databases, and then execute algorithms with this data.
- A main constraint with this strategy is that the databases sometimes cannot be moved to other network sites due to data-security, size, and privacy or data-ownership considerations.

In addition to the above mentioned reasons, it may happen that for some huge databases it may not be feasible to be stored and processed at one computer site.

Strategy–2: Pipelining

This strategy depends on executing the learning system on the first site to generate learning rule, then move these rules to second site and do learning to generate new rules, and so on until we reach the final site, this strategy has a lot of drawbacks that make it not valid. These drawbacks may be summarized as follows:

- It is a sequential learning strategy.
- Slow, inefficient, and may not work properly for massive data sets. The learning systems by this method is slow when it is used to learn from very large data sets, take more time in this process because it must be executed site by site, and carry the rules also from site to site until we reach the last site then transfer the resulted rules to the main site for execution. So, this process takes more time, take into consideration that the learning time is the summation of learning times of all site.
- Contradiction may arise in the generated rules. This is because that the learning process in this way cannot be incremental, because only the generated rules are moved to the next site without the datasets themselves which are needed for incremental learning.

Strategy–3: **Parallelism**

The learning strategy here is to execute the learning process on all locations separately and simultaneously, and generate the rules at each site, then merge all the rules of all sites in the main site and resolve the contradiction between them, which has the following characteristics that make it the most suitable strategy for learning:

- This strategy results in a fast learning process with high performance. This is because the learning system is executed at all sites simultaneously.
- The learning process in this strategy is incremental; rules can be refined by modifying their conditions; they do not need to be generated from scratch in each learning loop.

The third strategy seems to be the most adequate among other strategies for learning from a distributed database, so it will be adopted in our system.

The architecture of the rule discovery system developed is depicted as shown in Figure 1. The discovery system uses the JDBC API to communicate with the distributed database management systems (DDBMS). When a data mining session is started the system in every location sends meta-data queries to the DBMS connected. After the user selects a set of tables, the target table, and the target attribute, the data mining

process starts at all locations in parallel, during which the system sends a number of SQL queries to the DDBMS. SQL queries sent to the database management system are generally needed to build valid rules.

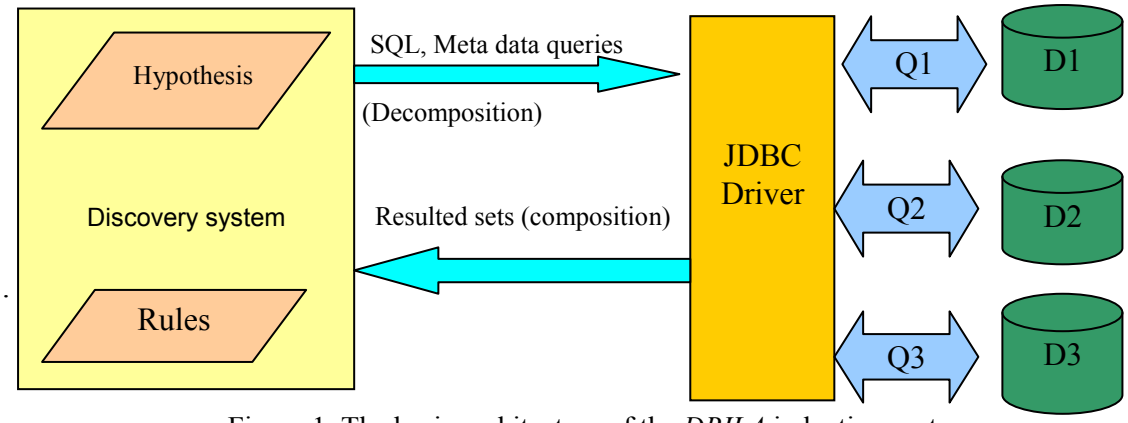

Figure 1. The basic architecture of the *DRILA* induction system.

In order to reduce the complexity of communication between the rule discovery system and the DDBMS, the information about covered objects and the discretized columns are both stored in temporary tables in the DDBMS rather than in the internal data structures in the rule discovery system side. It was also decided to use these temporary tables for performance reasons .

The temporary table 'covered' has two columns named 'id' and 'mark'. Each time the rule discovery system starts processing a new class, inserting a new row for each object belonging to the current class reinitializes this table. The 'id' field is given the value of the primary key and the 'mark' field is set to zero. When a new rule is generated, the 'mark' fields of the rows that refer to the objects covered by the new rule are changed to one .

2.3 The DRILA Algorithm

This algorithm is supposed to work in parallel way; i.e. must run in all locations simultaneously. It consists of two main phases:

Phase 1: Parallel learning: As learning would be done asynchronously.

Phase 2: Incremental learning: The rules resulted at this step are called *'selected rules'*. These selected rules should be distributed to other locations without repetition to reduce the effort, eliminate the contradiction and save consistency.

Depending on these two main phases, the detailed steps of DRILA are discussed in the following steps:

Step 1: Preparation of data sets

This includes the reconfiguration and normalization for datasets across all sites of the distributed database.

Step 2: Parallel learning

The parallelism strategy of learning is used here. The following steps; which are much similar to the steps of RILA algorithm; are executed in each location simultaneously to generate rules:

1- Constructing hypotheses:

The system sends meta-data queries to the connected DBMS to retrieve columns and foreign/primary keys information for the selected tables (target tables) from the database in the location. As soon as this information is retrieved, the system then has the complete schema description of the training data. The process in this step; as depicted in Figure 1 has the following two iterative steps:

i- The system sends SQL queries to this part of database.

ii- The results of these queries are analyzed to generate new hypotheses.

2- Refining, pruning and evaluating hypotheses:

After building the initial hypothesis, it is refined by adding new conditions and extending by adding new rules, and then some kind of heuristics are used to minimize the number of rules before they are processed by a phase called *'pruning*

heuristics'. In order to adopt a pruning method one needs to have a measure that can show the degree of possibility of a hypothesis and its possible extensions to become a credible rule by phase called *'hypothesis evaluation'*.

3- Rule selection:
Use the generated (selected) hypothesis with the highest score to generate rules, after a new rule has been selected, the system removes the examples covered by a new rule from the active search space, by using a temporary table in relational database to store the identifier of the examples covered by the selected rules which is important for keeping the data in its original from during the learning process, and implementing the '*effective cover'* used for avoiding redundant rule selection.

4- Traversing a relational schema:
'Rule selection' is repeated P times, deepening on the predefined parameter P, after first rule selected and before new rule asserting, DRILA algorithm checks whether examples covered by the candidate rule are not covered by the previously selected rule(s), so if there are examples not covered yet by previously selected rules then the candidate rule is asserted as new rule in the input rule set.
After *rule selection* is completed and there still objects not covered by the selected rules, then the initial hypotheses are rebuilt.

5- After level 1 is completed; i.e. generating all the rules with one condition; the algorithm moves to level 2; i.e. rules with two conditions; by firstly refines the best n hypotheses generated in level 1.

6- Perform rule selection as described in previous steps on level 1 to select rules for level 2.

7- Repeat these steps until the system reaches to the level m, determined by the parameters m, and all learning rules for this site are generated.

Step 3: Merging the rules and incremental learning

After finishing the parallel learning process indicated in step 2 at all locations and generating subsets of rules R_i, i=1…n in all sites, the system then collects R_i, i=1…n for all sites S_i, i=1…n into a main set of rules called R, and saves it into a file that can later be used in a prediction task. Rules in R should be contradiction free, so if a contradiction exists, it must be resolved immediately as part of the incremental learning.

The relationship between the steps of DRILA is summarized in Figure 2 for processing training examples of a single class. The database schema is treated as a graph where nodes represent relations (tables and edges represent foreign keys). The schema graph is searched in a breadth-first search manner starting from the target table. While searching the schema graph, the algorithm keeps track of the path followed; no table is processed for the second time.

Initial hypotheses are composed of only one condition at each location for each column except the foreign and primary key columns and the class column, i.e. the target attribute. If the current table is the target table then the algorithm uses a simplified its version.

The algorithm also needs to know about the frequency of the hypotheses in classes other than the current class. Similarly, for the target table, the algorithm uses a simplified its version.

After the initial hypotheses are generated they are sorted based on the output of the ILA hypothesis evaluation function, which shows how a hypothesis satisfies the conditions for being a valid rule. If any of the hypotheses can be used for generating a new rule then the one with the maximum score is converted to a new rule and the objects covered by the new rule are marked in the temporary table 'covered'. After the rule selection processes if some rules were selected but there are still objects not yet covered, then the initial hypotheses are rebuilt using only the objects that are not covered by the rules already generated. If no new rule can be generated then the hypotheses refinement step is started.

Refinement of a distributed relational hypothesis means extending the description of the hypothesis. It results in a new selection of objects that is a subset of the selection associated with the original hypothesis .Similar to the initial hypotheses build case, to extend a hypothesis, the schema graph is searched; starting from the target table, by following the foreign key relations between tables, here the hypothesis is the hypothesis object being refined. The object has two methods to help SQL construction processes. The table list method returns the list of the tables to which the features in the hypothesis refer, plus the tables that connect each

feature to the target table. The join list method returns the list of join conditions for the features in the hypothesis plus the list of join conditions to connect each feature to the target attribute. Figure 3 shows the flowchart of the simplified DRILA for processing examples of a single class at single location.

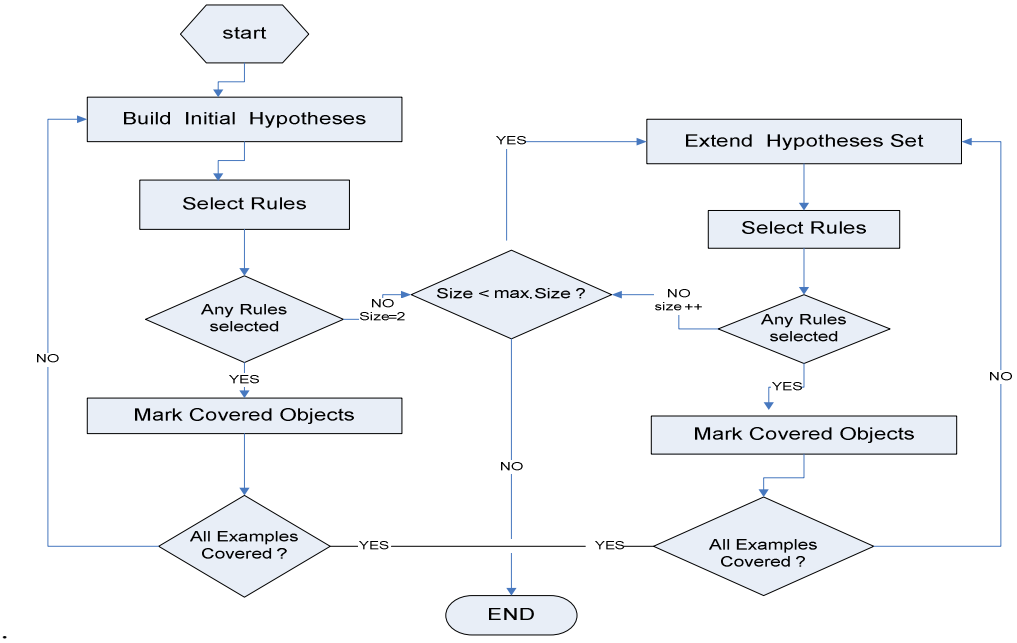

Figure 3: The simplified DRILA for processing examples of a single class at single location.

3 Experiments

3.1 Experiments on the Genes Data Set

A set of experiments are conducted using the genes dataset of KDD Cup 2001 [15] that distributed over three locations with different sizes regarding to the utilization at each site (location). There are three tables in the original genes dataset. One table (interaction) specifies which genes interact with which other genes. The other table (gene) specifies a variety of properties of individual genes. The gene table has information about 862 different genes. The third table (Composition) specifies the structure of each gene. There could be more than one row for each gene. The attribute gene_id identifies a gene uniquely (primary key). Tests have been conducted to generate rules for the localization attribute. Because the discovery system requires the target table to have a primary key, the schema has been normalized as shown in Figure 4.

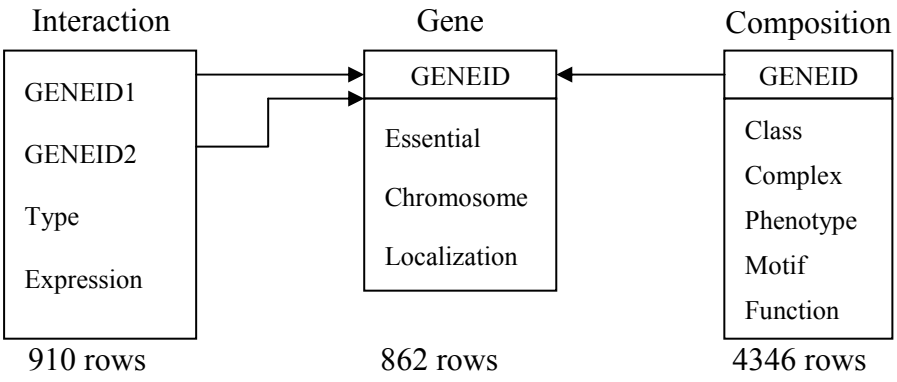

Figure 4. Schema of the KDD Cup 2001 genes data after normalization.

The dataset has one numeric attribute and several attributes with missing values .The numeric attribute, 'expression', in the interaction table was

divided into 20 bins using the class-blind binning method. Missing attribute values were ignored.

The results of the experiments are presented in Table 1 and Table 2. In the experiments, we change the parameter "Maximum number of hypotheses to be extended" to during 5 experiments from 1 to 5, while we fixed the other parameter as max rule size m= 3, F-measres f =0.01 and the penalty factor pf=2, level of recursive optimization l=100**.**

In Table 1 which reflects the results of learning centralized relational (merge) database, the first row shows the training time (seconds) which will be increased as the Maximum number of hypotheses to be extended increased, also the number of rules, number of conditions, while the training set accuracy is good and fixed during these experiments, but the training set coverage increases by small amount.

Table 1. Test results on the gene data set for (level of recursive optimization) l=100
Where: (Max rule size) m =3
(F-measures) f =0.01
(Penalty factor) pf = 2

(Max # of hypo. to be Extended) n	*n=1*	*n=2*	*n=3*	*n=4*	*n=5*
Training time (seconds)	*32*	*49*	*70*	*85*	*108*
Number of rules	*72*	*78*	*88*	*88*	*92*
Number of conditions	*93*	*106*	*131*	*129*	*139*
Training set accuracy	*93*	*93*	*93*	*93*	*93*
Training set coverage	*64.7*	*65.89*	*67*	*67*	*67.87*

In Table 2 which reflects the results of learning distributed relational database on three sites with same size, the first row shows the training time (seconds) which will be increased as the Maximum number of hypotheses to be extended increased, also the number of rules, number of conditions, while the training set accuracy is good and fixed during these experiments, but the training set coverage increases by small amount.

A comparison between the results in Table 1 and Table 2 shows that the training time, number of rules, and number of conditions are increased per the increasing of the maximum number of hypotheses to be extended, also according to the training accuracy and training coverage. But we observe that the training accuracy and training coverage in learning from distributed relational database is more superior efficient than that what we have in learning from centralized relational database.

Table 2. Test results on the gene data set for (level of recursive optimization) l=100
Where: (Max rule size) m =3
(F-measures) f =0.01
(Penalty factor) pf = 2

(Max # of hypo. to be Extended) n	**N = 1**			**N = 2**			**N = 3**			**N=4**			**N = 5**		
	loc A	Loc H	Loc G	loc A	Loc H	Loc G	loc A	Loc H	Loc G	loc A	Loc H	Loc G	loc A	Loc H	Loc G
Training time (seconds)	42	13	*30*	*59*	*23*	*75*	*133*	*33*	*55*	*139*	*41*	*89*	*370*	*52*	*89*
Number of rules	52	45	*43*	*55*	*51*	*44*	*60*	*55*	*45*	*66*	*55*	*45*	*70*	*56*	*45*
Number of conditions	70	53	*53*	*77*	*69*	*55*	*85*	*78*	*58*	*100*	*77*	*58*	*108*	*78*	*58*
Training set accuracy	92	95	*96*	*93*	*95*	*96*	*93*	*95*	*96*	*93*	*95*	*96*	*93*	*95*	*96*
Training set coverage	45	79.9	*85.6*	*46.5*	*82.64*	*86.25*	*48.23*	*84.38*	*86.6*	*51.06*	*84.38*	*86.6*	*52.48*	*84.03*	*86.6*

The results in the last row of Table 2 indicate that the discovered rules have about 20% more prediction accuracy. The reason behind the high performance noted in the table, lies in the present system's capability to read the relational information between genes defined by distributed database through many sites (three sites in our case).

3.2 Experiments on the Mutagenesis Data Set

Tests have also been conducted on the mutagenesis data set contains descriptions of molecules. The characteristic to be predicted is their mutagenic activity represented by the attribute label in the table molecule with 188 instances in it [16], that normalized distributed over three sites by average number of molecules about 62 at each location related logically to other tables of the database which are "bond" and "atom" as shown in Figure 5. The tasks to predict function and label for each of the molecules in the test set. The experiments in this study selected the label task. There are 2 labels and six additional attributes intrinsic to molecule (molecule-id, log-mut, logp, lugmo, ind1, ind2 and label) and two attributes concerning the interactions between genes. Figure 5 shows the schema of the mutagenesis data set used in the experiments

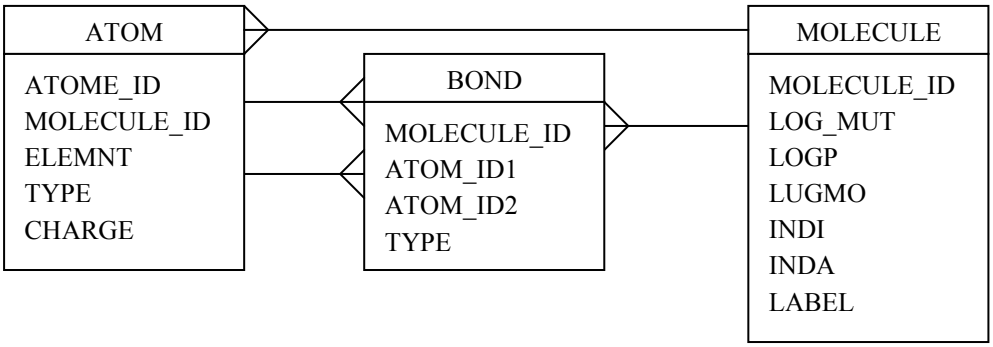

Figure 5 schema of the mutagenesis data set.

In the experiments, the parameter *m* (maximum size for rules) was set to 3, while the penalty factor (parameter *pf*) was selected as 2.

The results of the experiments using centralized database are shown in Tables 3, the parameter *m* was set to 3, the parameter *f* was set to 0.01, and the penalty factor was selected as 2. Table 3 shows the results of the experiments. The parameter *l* was selected as 100 (as explained above, the parameter *l* determines the level of recursive optimization to find a minimal rule set). For this reason, the experiments needed more time than the previous experiments that did not use the parameter *l*, (*l*=0). Using a large *l* value improved the accuracy of the results; however, the coverage of the rules decreased.

The results of the experiments on mutagenesis data set; that is distributed over three sites with same size of the database in every site, using RMI are shown in Tables 4. Using the same identical parameters that used in experiments on centralized database, parameter m was set to 3, the parameter *f* was set to 0.01, and the penalty factor was selected as 2. As shown in Table 4, the first row shows the training time (seconds) which is increased as the Maximum number of hypotheses to be extended increased, also the number of rules, number of conditions, while the training set accuracy is good and fixed during these experiments, but the training set coverage increased by small amount.

Table 3. Test results on centralized mutagenesis data set for (level of recursive optimization) *l*=100
Where (Max rule size m) =3
(F-measures, *f) =0.01*
(Penalty factor pf) = 2

(Max # of hypo. to be Extended) n	***n=1***	***n=2***	***n=3***	***n=4***	***n=5***
	DRILA	**DRILA**	**DRILA**	**DRILA**	**DRILA**
Training time (seconds)	*4.90*	*5.88*	*7.77*	*7.94*	*8.74*
Number of rules	*6*	*6*	*6*	*6*	*6*
Number of conditions	*7*	*7*	*7*	*7*	*7*
Training set accuracy	*96%*	*96%*	*96%*	*96%*	*96%*
Training set coverage	*96.85%*	*96.87%*	*96.91%*	*96.93%*	*96.95%*

Table 4 Test results on the mutagenesis data set for (level of recursive optimization) *l*=100
Where:
(Max rule size) m =3,
(F-measures), *f =0.01*
(Penalty factor) pf = 2

(Max # of hypo. to be Extended) n	***N=1***			***N=2***			***N=3***			***N=4***			***N=5***		
	loc A	*Loc H*	*Loc G*	*loc A*	*Loc H*	*Loc G*	*loc A*	*Loc H*	*Loc G*	*loc A*	*Loc H*	*Loc G*	*loc A*	*Loc H*	*Loc G*
Training time (seconds)	*4.97*	*6.12*	*6.17*	*4.97*	*6.127*	*5.06*	*6.35*	*7.51*	*6.35*	*5.58*	*7.4*	*6.35*	*6.43*	*7.5*	*8.01*
Number of rules	*3*	*3*	*3*	*3*	*3*	*3*	*3*	*3*	*3*	*2*	*2*	*4*	*2*	*2*	*4*
Number of conditions	*5*	*3*	*3*	*5*	*3*	*3*	*5*	*3*	*3*	*2*	*2*	*6*	*2*	*2*	*6*
Training set accuracy	*99%*	*92%*	*89%*	*99%*	*92%*	*89%*	*99%*	*92%*	*89%*	*98%*	*92%*	*90%*	*99%*	*92%*	*90%*
Training set coverage	*34.04*	*30.3*	*93.5*	*34.0*	*30.32*	*93.5*	*34.0*	*30.3*	*93.5*	*33.5*	*30.3*	*96.77*	*33.51*	*30.32*	*96.77*

4 Conclusions

In this paper, a distributed relational version called distributed relational inductive learning algorithm (*DRILA)* with a new rule search and selection strategy has been developed depending on an existing propositional relational learning algorithm called *(RILA)*, with two rule selection strategies; the *select early strategy* and the *select late* strategy which requires more learning time than the *select early strategy* but is more effective in finding the most efficient rule sets. In *DRILA* the *select* strategy, rule selection is performed after the hypothesis search process is completed. Three different pruning heuristics were used to control the number of hypotheses generated during the learning processes. Experimental results are presented on two data sets; the genes data set was used for the first time in the KDD Cup 2001 competition [15], and the mutagenesis data set on the mutagenic activity of molecules represented by the attribute label in the table molecule [16]. The system has several parameters to let users customize the algorithm execution for their specific goals in a given learning task.

Unlike many other relational learning algorithms, the *DRILA* algorithm does not need its own copy of distributed relational data to process it. This is important in terms of the scalability and usability of the distributed relational data mining solution that has been developed. The architecture proposed can be used as a framework to upgrade other propositional learning algorithms to relational learning.

This work has extended the *RILA* algorithm (which was designed for discovering rules from single centralized relational database) to the distributed relational domain, where complex objects can efficiently be represented using multiple tables (stored in many sites or locations over the network and managed by a distributed relational database management system). The system has been designed considering the efficiency of the learning processes in each site (location) related to the centralized location. For example, it avoids redundant hypotheses generation, i.e., each hypothesis is generated only once. Also, the bottom-up strategy used during rule specialization saves computation time by testing only combinations of features that exist in the training data. Our experience in adapting a propositional learning algorithm to the distributed relational domain can be useful for similar projects in the future. This experience also contributes to the general knowledge of distributed relational learning as this research describes its own approach to distributed relational learning which is different from other approaches in the ways described above.

The following features have been identified to improve the system's ability to mine distributed relational data:

- Techniques to handle the missing attribute values.
- In order to improve scalability of the system, names of the temporary tables used for storing discretization and coverage information can be annotated by a unique identifier of the learning process. This allows concurrent learning processes to use the same data without interfering with each other.
- Current Discretization strategy is based on the Weka library. This solution requires the numeric columns data to be transferred to the client side once when the discretization tables are created at the beginning of a data mining job. Generally numeric information does not require large volumes compared to text data. However, this strategy can be improved to remove the data transfer.
- Saving output rules in distributed relational databases, possibly using an object distributed relational mapping tool (such as Hibernate).
- Constructing a graphical representation of the input schema at the beginning of the learning process so there will be no need to search the foreign keys graph (schema graph) each time during hypothesis construction processes.

References:

[1] Knobbe, A.J., Blockeel, H., Siebes, A., Van der Wallen, D.M.G.: relational Data Mining, In *Proceedings of Benelearn'99*, (1999).

[2] Leiva, H., and Honavar, V.: Experiments with MRDTL—A relational Decision Tree Learning Algorithm. In Dzeroski, S., Raedt, L.D., and Wrobel, S. (editors): *Proceedings of the Workshop on Multi Relational Data Mining* (MRDM-2002), University of Alberta, Edmonton ,Canada, (2002) 97-112.

[3] Crestana-Jensen, V. and Soparkar, N.: Frequent Item-set Counting across Multiple Tables .*PAKDD 2000*, (2000) 49-61.

[4] Wrobel, S.: An Algorithm for Distributed Relational Discovery of Subgroups, *Proceedings of PKDD'97*, Springer-Verlag, Berlin, New York, (1997).

[5] SRS-Relational White Paper, Working with relational databases using SRS, LION

Bioscience Ltd. http://www.lionbioscience.com/solutions/products/srs.

[6] Elmasri R. and Navathe S., *Fundamentals of Database Systems*, Benjamin/Cummings, Redwood City, CA, Second edition, (1989).

[7] Silberschatz A., Korth H., Sudarshan S., *Database System Concepts*, Fourth Edition, McGraw-Hill Companies Inc., (2002).

[8] Neville J. and Jensen D. "Supporting relational knowledge discovery: Lessons in architecture and algorithm design", *Proceedings of the Data Mining Lessons Learned Workshop*, Nineteenth International Conference on Machine Learning, (2002).

[9] Muggleton S. (editor), *Inductive Logic Programming*, Academic Press, 1992.

[10] Quinlan J. R., "Learning logical definitions from relations", *Machine learning*, 5, pp. 239-266, (1990).

[11] Tolun, M. and Abu-Soud, S.: ILA: An Inductive Learning Algorithm for Rule Extraction, *Expert Systems with Applications*, 14(3), (1998) 361-370.

[12] Tolun, M., Sever, H., Uludag., M. and Abu-Soud, S.: ILA-2: An Inductive Learning Algorithm for Knowledge Discovery, *Cybernetics and Systems: An International Journa*l ,Vol. 30, (1999) 609-628.

[13] Uludag M., Tolun M and Etzold T., "A multi-relational rule discovery system", *Proceedings of Eighteenth International Symposium on Computer and Information Sciences*, Antalya, Turkey, (2003).

[14] Uludag M.: *Supervised Rule Induction for Relational Data*, PhD Dissertation, Eastern Mediterranean University, Cyprus, (2005).

[15] Cheng, J., Krogel, M., Sese, J., Hatsiz, C., Morishita, S., Hayashi, H. and Page, D.: KDD Cup 2001 Report, *ACM Special Interest Group on Knowledge Discovery and Data Mining* (SIGKDD) Explorations, Vol. 3, issue 2, (2002).

[16] King R., Muggleton S., Srinivasan A., and Sternberg M., "Structure-activity relationships derived by machine learning: the use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming", *Proceedings of National Academy of Science USA* 9, 93 (no. 1): (1996) pp. 438–442.

[17] L. Dehaspe and L. De Raedt, "Mining association rules in multiple relations", *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, 1297, Springer-Verlag, (1997) , pp.125–132.

[18] Clare A., Ph.D. Thesis, "*Machine learning and data mining for yeast functional genomics*", University of Wales, Aberystwyth, U.K, (2003).

[19] David Meyer, Friedrich Leisch, and Kurt Hornik. The support vector machine under test. *Neurocomputing* 55(1-2): 169-186, (2003)

[20] Corinna Cortes and V. Vapnik, "Support-Vector Networks", *Machine Learning*, 20, (1995).

[21] Basak J. and Kothart R., A Classification Paradigm for Distributed Vertically Partitioned Data, *Neural Computation*, Vol. 16, No. 7, (2004) Pages 1525-1544.