MRDTL: A multi-relational decision tree learning algorithm

by

Héctor Ariel Leiva

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:

Vasant Honavar (Major Professor)

Shashi Gadia

Drena Dobbs

Iowa State University

Ames, Iowa

2002

Graduate College

Iowa State University

This is to certify that the master's thesis of

Héctor Ariel Leiva

has met the thesis requirements of Iowa State University

_____

Major Professor

_____

For the Major Program

To my wife

Table of Contents

ACKNOWLEDGEMENT

I would like to express special thanks to Dr. Vasant Honavar for his help, guidance and mentoring; also to the members of the Artificial Intelligence Research Laboratory at Iowa State University, for the discussions and friendship; and to my committee members, Dr. Shashi Gadia and Dr. Drena Dobbs, for their help.

I am also grateful to my family that always believes in me.

Finally, all my thanks to my beloved wife Marivé for her unconditional love, help, support, and patience in this endeavor and always.

Much of this work was supported in part by grants from the National Science Foundation (0087152, 9972653), the Carver Foundation, Pioneer Hi-Bred, Inc. and the Fulbright Commission scholarship program.

ABSTRACT

Many real-world data sets are organized in relational databases consisting of multiple tables and associations. Other types of data such as in bioinformatics, computational biology, HTML and XML documents require reasoning about the structure of the objects. However, most of the existing approaches to machine learning typically assume that the data are stored in a single table, and use a propositional (as opposed to relational) language for discovering predictive models. Hence, there is a need for data mining algorithms for discovery of a-priori unknown relationships from multi-relational data.

This thesis explores a new framework for multi-relational data mining. It describes experiments with an implementation of a Multi-Relational Decision Tree Learning (MRDTL) algorithm for induction of decision trees from relational databases based on an approach suggested by Knobbe et al., 1999. Our experiments with widely used benchmark data sets (e.g., the carcinogenesis data) show that the performance of MRDTL is competitive with that of other algorithms for learning classifiers from multiple relations including Progol (Muggleton, 1995) FOIL (Quinlan, 1993), Tilde (Blockeel, 1998). Preliminary results indicate that MRDTL, when augmented with principled methods for handling missing attribute values, is likely to be competitive with the state-of-the-art algorithms for learning classifiers from multiple relations on real-world data sets drawn from bioinformatics applications (prediction of gene localization and gene function) used in the KDD Cup 2001 data mining competition (Cheng et al., 2002).

1. INTRODUCTION

Brief introduction to Knowledge Discovery in Databases (KDD) and data mining is given together with the main motivations for (multi)-relational learning.

1.1. Knowledge Discovery in Databases and Data Mining

Advances in information technologies are making possible for the scientific and industry communities to acquire, and store increased volumes of data in digital form, even more data than they can really process. This fact gave arise to a hybrid research field called Knowledge Discovery in Databases (KDD). Since its beginning, the research made in this field has been vast and is continuously growing.

Through the literature in the subject, the terms KDD and Data Mining sometimes are used indistinctly. A more recent convention mentioned in (Blockeel, 1998) establishes that the process of knowledge discovery actually consists of three subtasks. The first task is to adapt the original format of the data to fit the input format of the data mining algorithm (*Pre-processing of the data*). Once formatted the data, one or more algorithms must be applied to extract patterns, regularities or general laws from the data, this is the phase properly called *Data Mining*. Once the results of the data mining process are obtained they may require to be translated to a more intelligible format. This last stage is known as *post-processing of the results*.

Essentially, KDD is concerned with the nontrivial identification and extraction of valid, novel, potentially useful, and ultimately understandable knowledge from (large) databases, and data mining is the central step in this process (Dzeroski and Lavrac, 2001).

Data mining is situated at the intersection of several scientific domains. It combines techniques from statistics, databases, computer graphics, artificial intelligence and machine learning.

Data Mining algorithms can be classified into two broad categories: predictive or descriptive approaches. In predictive data mining, the aim is to induce a hypothesis that correctly classify all the given examples and can be used for classification of future, yet unseen, instances. The most popular example of predictive technique is decision tree induction (e.g., Quinlan's C5 (Quinlan, Rulequest)). On the other hand, the goal in descriptive data mining is to characterize as much as possible, by finding patterns regularities, the given examples. Discovering of association rules is one example of this class of data mining techniques (De Raedt *et al.*, 2001).

There exist other analytical tools that explore data interrelationships, for instance OLAP (On-Line Analytical Processing). But the difference between KDD and these tools is in the approach they use. Many of the analytical tools available support a verification-based approach, in which the user hypothesizes about specific data interrelationships and then uses the tools to verify or refute those hypotheses. The effectiveness of this approach relies mainly in the knowledge of the domain in question. On the other side, the goal of data mining is to find patterns of influence. For instance, how does certain attributes affect the sales of a certain business. Efforts have been made to combine existing analytical tools such as OLAP and Data Mining techniques (e.g. (Information Discovery)).

1.2. Relational Learning

Traditionally, research in KDD has focused mainly on attribute-value learning where each example or instance can be characterized by a fixed set (tuple) of attributes for which values are given. The data set in this case can be viewed as a table (relation) where each row corresponds to an instance and each column to an attribute. The hypothesis language is propositional logic, where propositions are of the form "Attribute $\oplus$ value" where $\oplus$ is an element of a predefined fixed set of operators such as $\{<, \geq, =\}$ (Blockeel, 1998).

Most existing databases in different domains are not stored as a single relation, but as several relations for reasons of nonredundancy and storage and access efficiency. An

instance in a relational database can consist of several records fragmented across several tables. Furthermore, we can see the structure of a relational database as a way of representing some knowledge about the domain in question that adds to the information represented by the tuples themselves. Initially, attribute-value learning can not represent this kind of background knowledge or find relationships between different records from one or more tables. Although in principle, it is possible to reconstruct a single relation from multiple tables; such an approach is fraught with many difficulties in practice. Consequently, the task of learning from relational data or multi-relational learning has begun to receive significant attention in the literature (Blockeel, 1998, De Raedt, 1998, Knobbe *et al.*, 1999a, Knobbe *et al.*, 1999b, Friedman *et al.*, 1999, Koller, 1999, Krogel and Wrobel, 2001, Getoor, 2001, Kersting and De Raedt, 2000, Kersting *et al.*, 2000, Pfef01, Dzeroski and Lavrac, 2001, Dehaspe and De Raedt, 1997, Dzeroski *et al.*, 2001, Jaeger, 1997, Karalic and Bratko, 1997).

When using relational databases containing multiple relations, it is not always necessary to use a relational data mining algorithm. Luc De Raedt in (De Raedt, 1998) outlines how a special case of relational learning can be transformed into attribute-value learning. But in the general case, if we want to use traditional (propositional) learning algorithms then the multiple tables that a relational database consist of have to be collapsed into a single table. There are two general ways of transforming a multi relational database into a single relation.

1. Create a universal relation which involves the join of all the tables to form a single one. There are some potential problems to this approach:
    - The resulting universal relation can be extremely large and impractical to handle.
    - Increased data redundancy resulting in a considerable increase in the size of the data set.
    - The data duplication resulting from flattening the data may introduce statistical skew (Getoor, 2001).
    - Attribute value learning becomes much more inefficient than first order (relational) learning on some kinds of problem (De Raedt, 1998).
2. Transform a multiple relation database into a single relation by creating new attributes in a central relation that summarizes or aggregate information from other tables. This approach shares similar drawbacks with the first method plus an additional one.

Although the new attributes do not add more information about the examples in the database, it may not be easy to find appropriate ones. This stage may require a great amount of domain understanding and it is part of the knowledge discovery process. One way to deal with this is to let the attribute value learner itself to come up with good attributes (feature construction) or to enlarge the hypothesis space by allowing tests involving multiple attributes (e.g. attribute1 = attribute2) (Blockeel, 1998).

Finally, there are domains where reasoning about the structure of the objects and relations between them is inherently required (De Raedt *et al.*, 2001). For example, Blockeel identifies a chemical database that describes molecules (molecules table) by listing the atoms (atoms table) and bonds (bonds table) that occur in them and using Mendelev's periodic table of elements as background knowledge (mendelev table), where learning from multiple relations is desirable, and where joining the relations into a single one is not feasible unless a multiple-instance learner is used (Blockeel, 1998). A multiple-instance learning task is one where each training example is represented by multiple instances (or feature vectors) (Dietterich *et al.*, 1997). If we do not use a multiple-instance learner, then all the atoms with their characteristics a molecule consists of have to be stored in one tuple with indefinite number of attributes. This is also applicable to the Mutagenesis database (Srinivasan *et al.*, 1996) described in more detail in Chapter 4.

Thus, there is a clear need for data mining techniques that take into consideration the relational representation of the domain of the task to be learnt.

Most of the research done in this topic has been made in the field of Inductive Logic Programming (ILP), described in more details in the next chapter, with an extensive literature available. Most recently, other different techniques not based on logic formalisms (Getoor, 2001, Knobbe *et al.*, 1999a, Knobbe *et al.*, 1999b) and others based on logic but not strictly ILP such as (Kersting and De Raedt, 2000) have been proposed with relevant success.

A number of techniques proposed so far for relational data mining, in any of the previously mentioned fields, rely on their propositional counterparts. For example, the ILP systems Claudien, ICL, and Tilde are first order upgrades of propositional data mining algorithms that induce association rules, classification rules and decision trees respectively (De Raedt *et al.*, 2001); the approach to relational reinforcement learning proposed in

(Dzeroski *et al.*, 2001) is a combination of reinforcement learning and ILP; bayesian logic programs and probabilistic relational models are extensions of Bayesian networks to relational domains; finally the decision tree induction proposed in (Knobbe *et al.*, 1999b) is a clear extension of logical decision trees used in Tilde, which in turn is based on propositional decision tree induction algorithm. Basically all of them follow a similar methodology to that mentioned in (De Raedt, 1998) to upgrade attribute-value learning algorithms to cope with relational learning.

Figure 1.1 shows the relationships between propositional and relational learning algorithms previously mentioned. Next chapter expands on this taxonomy.

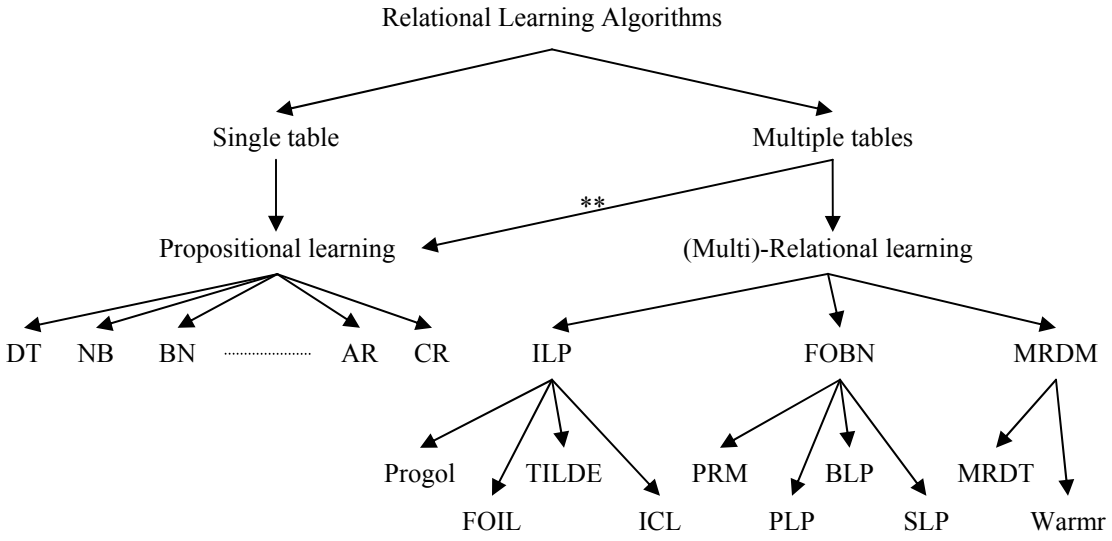


Fig. 1.1: Relationships between propositional and relational learning algorithms. Only the most representative algorithms of each class are shown.

References:

**: Proposionalization – join of tables
DT: Decision Tree
NB: Naïve Bayes
BN: Bayesian Networks
AR: Association Rules
CR: Classification Rules
ILP: Inductive Logic Programming
FOBN: First Order Bayesian Networks
MRDM: Multi-Relational Data Mining
FOIL: First Order Inductive Logic
TILDE: Top-down Induction of First-order Logical Decision Trees
ICL: Inductive Classification Logic
PRM: Probabilistic Relational Model
PLP: Probabilistic Logic Program
BLP: Bayesian Logic Program
SLP: Stochastic Logic Program
MRDT: Multi-Relational Decision Tree

1.3. Motivation

Barring a few notable exceptions (Getoor, 2001, Kerst00), most of the research in the subject has been based on ILP techniques. But the use of ILP engines in real world applications has been limited due to input specification, lack of efficiency while not taking into account relational databases issues and it inability to deal with noise and missing values in the data. Therefore another approach that benefits from the vast research done in databases (mainly about efficiency), knowledge discovery and inductive logic programming where the database end user does not have to deal with a new formalism such as logic but instead use the direct relational representation of the database in question is attractive (Chapter 2 explains how ILP can be adapted to cope with relational databases not in prolog format).

The multi-relational data mining framework proposed in (Knobbe *et al.*, 1999a) is a novel approach that exploits the semantic information in the database making use of the known Structured Query Language (SQL) to learn directly from data in a relational database. The same authors, in (Knobbe *et al.*, 1999b), introduce a general description of a decision tree induction algorithm, based on the multi-relational data mining framework and logical decision tree (Blockeel, 1998). To the best of our knowledge, there are no experimental results available concerning the performance of the algorithm for induction of decision trees from a relational database proposed in (Knobbe *et al.*, 1999b). Therefore, this work concentrates in a possible implementation of a multi-relational decision tree learner (MRDTL) based on this framework using Java as the programming language and Oracle as the backend relational database; and compares the performance of MRDTL with several other approaches on some representative multi relational data sets including those used in KDD Cup 2001.

The idea of shifting from ILP techniques to a search space consisting of database queries using SQL instead of logical clauses as the language bias and results from database theory as the next step in the relational data mining field had been already proposed in (Blockeel and De Raedt, 1997a). There, Blockeel *et al.* borrow techniques from ILP to tackle the problem of finding relationships between relations using relational algebra as the language bias. An equivalent algorithm using SQL is straightforward. That algorithm has not been implemented yet.

1.4. Organization of the Thesis

Chapter 2 discusses some of the most relevant approaches to relational learning that have been studied on different Computer Science's areas.

Chapter 3 describes in more detail the multi-relational framework to which the relational learning algorithm implemented belongs together with an explanation of the own characteristics of the system.

Chapter 4 details the relational databases used to test the performance of MRDTL, exposes results obtained with it and contrasts them with those results already in the literature for the same set of relational databases obtained with other techniques for relational data mining.

Finally, Chapter 5 concludes the paper and enumerates possible future extensions.

2. RELATIONAL DATA MINING APPROACHES

Three main directions can be distinguished in the area of relational data mining: 1) Inductive Logic Programming (ILP), where an extensive research has been made; 2) First order extensions of probabilistic models; and 3) Approaches that borrow ILP techniques but where the search space consist of database queries exclusively.

This chapter discusses these three main approaches, characterizing them from a relational database point of view and concludes enumerating a series of recent techniques as well as extensions to some relational algorithms previously proposed.

2.1. Inductive Logic Programming

As its name indicates, ILP is situated at the intersection of two important areasof Computer Science: Induction that is one of the main techniques used in several Machine Learning algorithms to produce models that generalize beyond specific instances and Logic Programming which is the programming paradigm that uses first order logic to represent relations between objects and implements deductive reasoning. The main representative of this paradigm is Prolog.

The initial focus of ILP was to develop algorithms for the synthesis of logic programs from examples and background knowledge (i.e. knowledge acquisition for some domain). More recent developments in ILP have considered classification, regression, clustering, and association analysis (Dzeroski and Lavrac, 2001). Thanks to its flexible and expressive way of representing background knowledge and examples, the field has also been expanded from the single-table case to the multiple-table one.

For clarification purposes, a small ILP example (taken from (Lavrac, 2002)) is given below.

Let

E^+= {daughter(Mary, Ann), daughter(Eve, Tom)} be the positive examples of the relation daughter;

E^-= {daughter(Tom, Ann), daughter(Eve, Ann)} be the negative examples of the same relation; and

B= {mother(Ann, Mary), mother(Ann, Tom), father(Tom, Eve), father(Tom, Ian), female(Ann), female(Mary), female(Eve), male(Pat), male(Tom), parent(X,Y) ← mother(X,Y), parent(X,Y) ← father(X,Y)} be the background knowledge,

where the relations *daughter*, *mother*, *father*, *female*, *male*, and *parent* have the common meaning. Then, the goal of this ILP problem is to learn the concept *daughter*. For predictive ILP, a solution could be the following synthesized clause:

daughter(X,Y) ← female(X), parent(Y,X).

or a set of definite clauses:

daughter(X,Y) ← female(X), mother(Y,X).

daughter(X,Y) ← female(X), father(Y,X).

Although induction is the inverse operation of deduction, the inductive process is more difficult than deduction. For deduction, we can use sound rules of inference but inductive inference involves unsound conjectures based on statistical support from data (Muggleton, 1995). Due to this, there is not a well established ILP engine yet as there exists in deductive logic programming with Prolog.

ILP has been one of the first approaches taken in relational learning and one of the most expanded. However its use in relational data mining solutions in KDD has been limited due to the existence of different ILP engines in terms of input specification (Knobbe *et al.*, 2001), the language bias they use and the limited attention they pay to relational database theory. For example, if we try to run the exercise presented in (De Raedt *et al.*, 2001) for the three ILP systems there described (Claudien, ICL and TILDE), or another example that is stored in a relational database, we will need to create a knowledge database if it is needed, code the examples using logical formalisms and adapt the input for each of the three systems (two of

them share the same input format specification). They do not exploit the capabilities of relational databases.

A number of different techniques have been proposed to cope with the drawbacks of ILP:

- In order to deal with the logic formalism and to unify the different input specifications of different ILP engines, (Knobbe *et al.*, 2001) propose the use of Unified Modeling Language (UML) as the common specification language for a large range of ILP engines. The idea is to replace the logic-based formalism to specify the language bias with a language that is easy to use, can be used by a range of ILP systems, and can represent the complexity of the problem in a clear and simple way. In that way, even the non-expert users will be able to model their problems and use a wide range of ILP engines, choosing that one that best fits their current needs.
- In (Blockeel and De Raedt, 1997a), three ways of bridging ILP and relational databases are presented. These are possible solutions to the problem of efficiency issues mentioned before. They are briefly described in order of adaptation of the ILP system to the database system.
  - The simplest one is pre-processing of the relational data to be transformed to Prolog syntax (an algorithm for this purpose is given in (Blockeel, 1998)).
  - The second possible bridge relies on the fact that some Prolog systems can be linked to relational databases. A relational database is available together with the Prolog knowledge base. Each time a literal *p(a, b)* has to be evaluated and the predicate *p* corresponds to a relation *P* in the relational database, Prolog has to open a connection and query the database to determine if the tuple *(a, b)* is in *P*.
  - The last bridge exploits the close relationships existent between first order logic and relational databases (a predicate is a relation between its arguments which are, in turn, attributes in relational databases) and between logical clauses and relational database queries. Therefore, entire logical clauses (not only a literal at a time as in the second approach) can be translated to SQL statements and submitted to the database to obtain statistics. Systems implementing this characteristic already exist. For example RDT/DB (Lindner and Morik, 1995) (cf.

(Blockeel and De Raedt, 1997a)) couples the ILP system RDT (Kietz and Wrobel, 1992) with the Oracle database system.

There exist several and different ILP engines and most of them can be placed within one of the two main logical approaches: *learning from entailment and learning from interpretations*. The first one is considered the classical setting, also called *explanatory ILP* (Blockeel, 1998) and most of ILP systems use this setting (e.g. RDT (Kietz and Wrobel, 1992), Progol (Muggleton, 1995), FOIL (Quinlan, 1993a), SRT (Kramer, 1996), Fors (Karalic and Bratko, 1997)). Learning from interpretations is also called *descriptive ILP* and more recently, ILP engines based on this setting have begun to appear (e.g. Claudien, ICL, and Tilde (De Raedt *et al.*, 2001)). An explanatory history of how learning from interpretations evolved from a descriptive technique to a predictive one is given in (Blockeel, 1998). Additional approaches to ILP can be found in (De Raedt, 1997).

The two main ILP approaches differ from each other in the way they represent the examples, the background knowledge and the way the final hypothesis is induced. In learning from interpretations each example e is represented by a separate Prolog program encoding its specific properties as sets of interpretations and the background knowledge B is given in the form of another Prolog program. The learning from entailment paradigm represents all the data together with the background knowledge as a simple Prolog program, there are no separations between examples.

A hypothesis in the learning from interpretations setting is a set of clauses such that only each positive example together with the background knowledge makes each clause in the set true. In notation, being E^+ and E^- the sets of all positive and negative examples respectively, a hypothesis H has to be found such that

$\forall e \in E^+ : H$ is true in $M(e \wedge B)$

$\forall e \in E^- : H$ is false in $M(e \wedge B)$

where $M(e \wedge B)$ is the minimal Herbrand[1] model of $e \wedge B$.

Whereas in the learning from entailment framework the induction problem is to find H such that the following entailment constraint holds

[1] A brief but useful review about some basic concepts in first order logic can be found in (De Raedt, 1997).

$\forall e \in E^{+} : H \wedge B$ entail e

$\forall e \in E \ : H \wedge B$ not entail e

These definitions are given in (Blockeel, 1998), a slightly different definition for learning from interpretations is given in (Blockeel and De Raedt, 1997a) where they do not distinguish between positive and negative examples.

In turn, within learning from entailment we can distinguish two principal approaches to the induction process: inverse deduction and a generalization of top-down induction techniques to the first-order case (Russel and Norvig, 1995). An example of the first case is Progol (Muggleton, 1995) that uses inverse entailment as a generalization and enhancement of previous techniques for inverse deduction and for the second approach FOIL (Quinlan, 1993a), one of the first ILP systems using top-down induction. But, in learning from interpretations top-down induction of hypothesis is the most commonly used technique.

Although it has been shown that learning from interpretations reduces to learning from entailment (De Raedt, 1997), which in turn reduces to learning from satisfiability (learning from partial interpretations, where missing values are considered), an increased interest has been focused toward learning from interpretations in several recent ILP systems (e.g. (Blockeel, 1998, De Raedt *et al.*, 2001). The latter is due to the fact that attribute value representations are a special case of this setting. But also, because many attribute value techniques exploit the independence of examples and the setting in question assumes also this independence through the separation of information between examples, then attribute value learning algorithms can be upgraded in a more trivial way to this setting than to learning from entailment as it has been shown by the systems Claudien, ICL and Tilde (De Raedt *et al.*, 2001). This setting has also inspired the shifting from a pure logical search space to one consisting exclusively of database queries in relational algebra or SQL-like language in (Blockeel and De Raedt, 1997a, Knobbe *et al.*, 1999a, Knobbe *et al.*, 1999b).

2.2. First Order Extensions of Bayesian Networks (BNs).

Traditional approaches in attribute-value learning and learning from interpretations settings mentioned before assume data independence in order to come up with a hypothesis explaining the examples. Probabilistic models, most specifically Bayesian networks (Pearl,

1991), differ from those approaches by specifying a probability distribution over a fixed set of typically discrete random variables that could be the attributes in an attribute-value setting. Thus, Bayesian nets are a probabilistic extension of propositional logic (Kersting and De Raedt, 2000). One of their most important features is their capacity of reasoning under uncertainty. But, probabilistic models inherit the same limitations as propositional logic when they are applied to relational data.

In order to address these limitations, several approaches for combining first order logic and Bayesian nets have been proposed. They have risen in different fields and under different circumstances. The most representatives are Probabilistic Logic Programs (PLPs) (Ngo and Haddawy, 1997) from logic programming; Relational Bayesian Networks (RBNs) (Jaeger, 1997) from model theory and Probabilistic Relational Models (PRMs) (Koller, 1999) that have their foundations in the database field. In spite of their different background, they seem to share a common kernel represented by Bayesian Logic Programs (BLPs) as shown in (Kersting and De Raedt, 2000). This last approach is a simplification and reformulation of PLPs, so we will concentrate on this last one instead of the former one.

2.2.1. Brief Introduction to Bayesian Networks

Bayesian networks consist of two components: a qualitative one and another quantitative. The qualitative part is represented by a directed acyclic graph (DAG) where each node represents a domain variable (generally called random variables), and each arc represents a probabilistic dependency between two variables. The probabilistic dependency is the quantitative part represented through a conditional probability distribution for each node. An example of a Bayesian network taken from (Friedman and Koller, 2001) is shown in Fig. 1.1. The format in that figure is somehow different to the usual one, but this is to agree with the format of BLPs given below.

A BN is a compact representation of probability distributions via conditional independence. A BN can compute the conditional probability for each value of a node given that values have been assigned to the other nodes. In this sense, BN classifiers can compute the conditional probability of a class value given the values of the other attributes. From the example in Fig. 1.1, where the conditional probability table for alarm node is given, we can

see that the alarm starting depends on the values of *burglary* and *earthquake*. For a more detailed description of BNs refer to (Jensen, 2001).

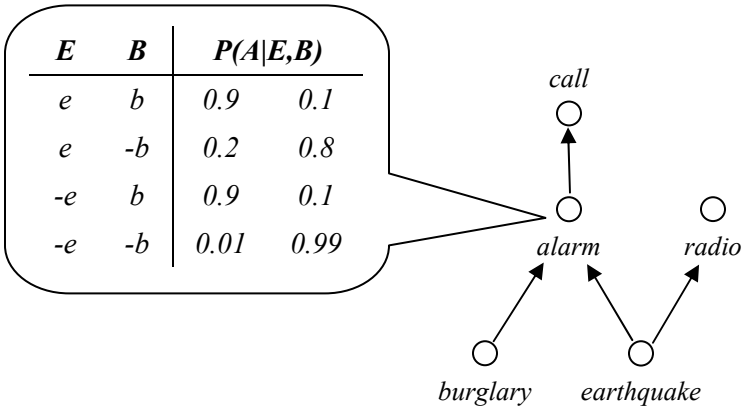


Fig. 2.1: Example of a quantitative component (dialog box) and the qualitative component (right) of a simple Bayesian network.

There are several motivations to the use of BNs for relational data mining. The main one was mentioned in the first paragraph of this section and refers to the fact that BNs can naturally represent relationship among attributes and this is clearly needed in relational domains, but also they can represent dependencies between data. At the same time, they can make decisions under uncertainty.

Another motivation to the use of Bayesian nets-based techniques for relational classification tasks is their intrinsic feature selection capacity based on the concept of *Markov blanket* of a node in the network. Basically, the *Markov blanket* of node *n* is the union of *n*'s parents, *n*'s children, and the parents of *n*'s children. For complete data, the random variables outside Markov blanket of node *n* can be deleted from the BN producing a smaller network. Making node *n* the attribute to be predicted in the classification process, all others nodes that do not belong to its Markov blanket can be deleted without affecting the classification accuracy of the network. This, together with information gain based feature filtering is a novel approach for attribute-value learning of high-dimensional data that has shown to be successful in several domains outperforming classical classification algorithms such as decision trees (Cheng *et al.*, 2002).

Finally, their structural representation makes the relationships between attributes easy to understand and can be modified by domain experts in order to obtain better predictive models or models with certain characteristics.

One of the drawbacks of BNs is that generally the domain of each of the random variables has to be finite. Thus, some discretisation algorithm must be used before learning the network and its parameters, sometimes loosing some important information.

2.2.2. Relational Bayesian Networks (Jaeger, 1997)

In RBNs the nodes are predicate symbols corresponding to every relation *r* of some vocabulary *S*, the values of these random variables are the possible interpretations of the predicates over some finite domain. The semantic of a RBN is a probability distribution over the relations of *S*.

The qualitative component of RBNs is represented by Probability Formulae (PFs) that employ combination functions as well as equality constraints. These PFs allow nested combination functions too and not only specify the CPT but also the dependency structure.

2.2.3. Probabilistic Relational Models (Getoor, 2001)

PRMs are an extension of BN to relational domain where the nodes of the network are the attributes of a relational database. The edges represent probabilistic dependences between attributes. These dependences between attributes are either true or false; they do not have certain probability of occurrence as in RBN. The semantic of a PRM is a probability distribution over the attributes and the relations that hold between them (Getoor, 2001).

Like traditional BNs, PRMs consist of two components: the structure of the network and the parameters (conditional probability distribution) associated with it. For learning PRMs, (Friedman *et al.*, 1999, Friedman *et al.*, 2001) extend the techniques for learning BNs to learn from structured data. The two components mentioned before are the ones that have to be learned. As it happens in BNs, the parameter estimation is the easiest task of both and the key here is the likelihood function (i.e. the probability of the data given the model). The higher this probability, the better the model to predict the data. For structure learning three components need to be defined: the hypothesis space (which structures the algorithm can consider); a scoring function to evaluate each hypothesis relative to the data; and the search

algorithm. This last one is a heuristic and the most used one is the greedy hill-climbing search.

2.2.4. Bayesian Logic Programs

This kind of first order extension of Bayesian networks is introduced in (Kersting and De Raedt, 2000, Kersting *et al.*, 2000) as a simplification and reformulation of PLPs, but also as a common framework for the previous mentioned approaches.

In BLPs, the qualitative part of the Bayesian net is represented by a set of Bayesian definite clauses. The difference between this kind of clauses and classical clauses is that each ground atom in a BLP represents a random variable (Kersting and De Raedt, 2000). The set of random variables corresponds to the least Herbrand model of this logical program, i.e. the set of all ground atoms that are logically entailed by it. The parents of a certain random variable *n* are all facts directly influencing *n*. An example is given in Fig. 1.2.

For a determined query, it is not necessary to compute the least Herbrand model of the complete logic program; we need to consider only those variables that influence the query result. A certain query (as in Fig. 1.2) asks for the probabilities of the clause of taking any of the values in its domain. The query ?- *alarm(james)* asks for the probabilities of *alarm(james) = true* and *alarm(james) = false*, for instance.

```
lives_in(james, yorkshire).
lives_in(stefan, freiburg).
neighborhood(james).
tornado(Yorkshire).
burglary(X) | neighborhood(X).
alarm(X)    | burglary(X).
alarm(X)    | lives_in(X,Y), tornado(Y).
```

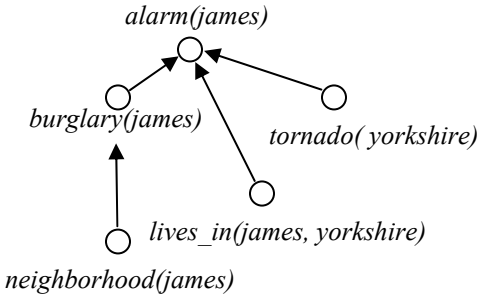


Fig. 1.2: example of a Bayesian logic program (left) and the corresponding Bayesian structure for the query ?- alarm(james). (right)

The quantitative part is as in BN represented by conditional probability tables and combining rules stored for each node or literal. A query can be answered by using any Bayesian net inference engine.

Figure 1.2 shows how a Bayesian net can be obtained from a logical program. An example of a Bayesian logic program obtained from a Bayesian network can be observed in Fig. 1.3. That program corresponds to the Bayesian net described in figure 1.1 of section 2.2.1 of this chapter.

```
burglary.
earthquake.
alarm      :- burglary, earthquake.
radio      :- earthquake.
call       :- alarm.
```

Fig. 1.3: Bayesian logic program corresponding to the Bayesian net given in the introduction to this section.

Algorithms for learning BLPs (the qualitative and the quantitative parts) have been proposed in (Kersting *et al.*, 2000) but have not been implemented yet. The authors of PRMs suggest that algorithms for learning their models can be adapted for BLPs too (Getoor, 2001).

2.2.5. Combining First-Order Logic and Probability Theory

Other approaches for combining first-order logic and probability theory (not only Bayesian networks) have been proposed, one of the most interesting is Stochastic Logic Programs (SLPs) (Muggleton, 1996). , Cussens, 1999, Muggleton 2000).

Stochastic Logic Programs (SLPs) are a generalization of stochastic context-free grammars (Lari and Young, 1990) and Markov nets (or undirected Bayes net) (Cussen, 1999). An SLP consists of a set of labeled clauses $p{:}C$ where $p \in [0,1]$, C is a first-order range-restricted definite clause (every variable in the head of C is the body too) and for each predicate symbol q the probability labels for all clauses with q in the head sum to 1. An SLP induces a probability distribution over the possible ground facts for a given predicate.

An algorithm for learning SLPs is described in (Muggleton, 2000). Consists of two steps (as in Bayesian networks):

1) Construction of the logic program (LP) with uniform probability labels for each clause and near maximal posterior probability with respect to the examples, and
2) Parameter estimation to increase the posterior probability with respect to the examples.

For the first part Muggleton used Progol4.5, i.e. the task of learning the structure of an SLP is based on ILP approaches. But step two uses a maximum likelihood approach, as most Bayesian approaches use, which is based on maximizing the posterior probability of the program.

2.3. Multi-Relational Data Mining

The term multi-relational data mining was first used by (Knobbe *et al.*, 1999a) to describe a new approach for relational learning. Later this term was accepted to describe the multi-disciplinary field that deals with knowledge discovery from relational databases and data which consists of complex/structured objects. In this section and next chapter, the term in question is used as proposed in (Knobbe *et al.*, 1999a).

The idea of shifting from a pure logical search space as in ILP to a search space consisting of database queries but borrowing techniques from ILP was firstly proposed in (Blockeel and De Raedt, 1997a), although for the best of our knowledge, that algorithm has not been implemented so far. That algorithm tries to find relationships between relations and it is given for relational algebra but a translation to a database query language such as SQL should not be difficult and would be more useful.

That first attempt was motivated by the fact that most of the ILP systems were not integrated with a relational database but rather with a deductive database such as a Prolog database, connections to relational databases had to be done by using the techniques mentioned in section 2.1. Therefore, systems that exploit relational databases' characteristics directly were needed.

The same group then proposed a multi-relational data mining (MRDM) framework in (Knobbe *et al.*, 1999a) that also borrows techniques from ILP but the search space consists of

SQL queries. This approach is detailed more precisely in the next chapter. Here, a brief introduction is given.

MRDM framework can support a range of relational data mining algorithms generalizations of common attribute-value learning algorithms. Knobbe *et al.* showed this by adapting the Warmr algorithm (Dehaspe and De Raedt, 1997) which is an ILP generalization of the Apriori algorithm (Agrawal *et al.*, 1996) to find association rules in multiple relations in (Knobbe *et al.*, 1999a), and also by adapting the first order logical decision tree induction proposed in (Blockeel, 1998) that is a first order upgrade of the common top-down induction of decision tree in (Knobbe *et al.*, 1999b).

One of the first drawbacks to appear when the search space of a relational learning algorithm consists of structures considering relational databases queries only is that SQL has lower expressive power than first order logical clauses. For instance, if a concept consists of a recursive definition in terms of one or several relations, it can not be represented in relational algebra or SQL. This should not be a problem for most of the existing relational databases in different domains because the expressive power of non-recursive languages such as SQL is enough to query those databases (Blockeel and De Raedt, 1997a).

Finally, SQL provides more possibilities than relational algebra and some different features than logic programming such as grouping, counting, etc. Then, it is expected that the search space be organized in a different way than the clausal logic space and the relational algebra space.

2.4. Other Recent Approaches and Extensions

When relational learning was introduced in the first chapter, the second way of transforming a multiple relations database into a single relation to which propositional learning algorithms can be applied to was the creation of new attributes in a central relation that summarizes or aggregate information from other tables in the relational database. Variants of this method have been used in systems such as LINUS (Lavrac *et al.*, 1991) and DINUS (Lavrac and Dzeroski, 1994) (successor of the previous one). These two systems are examples of a class of ILP methods that transform restricted ILP problems into attribute-value form through the technique called *proposionalization* and solve the transformed problem with a propositional

learning algorithm. These two systems have some limitations that do not allow them to tackle some problems that can be present in relational databases such as non-determinate relationships (a record in one table can have more than one corresponding record in another table).

More recently, (Krogel and Wrobel, 2001) introduce a new transformation-based ILP learner similar to the previous mentioned systems that can deal with non-determinate relationships, non-constrained literals and it is oriented to the business domain where the relational databases are often structurally simple, but large in size. This approach is called *Rel*y on *Agg*regation, *S*ometimes (RELAGGS) and has been applied with significant success to learning problems from the business domain (Krogel and Wrobel, 2001) but also from the biology domain (Cheng *et al.*, 2002). This feature construction method shows that if good attributes are constructed by the *proposionalization* algorithm, a feature-based learner can outperform relational learning techniques.

Continuing in the same trend, (Neville and Jensen, 2000) propose the use of a simple propositional Bayesian classifier in an iterative way for relational domains. The approach keeps the relational structure and the objects are "flattened" when is required by the algorithm. Inferences made about an object can assist inferences about related objects; therefore, inferences made with high confidence about some objects are fed back into the database and used for posterior inferences about possible related objects. Although simple Bayesian classifier is an attribute-value learner, by keeping the relational structure of the data helps to perform the feedback procedure into related objects.

In the framework of Probabilistic Relational Models, (Pfeffer, 2000) proposes an extension to this language in order to make PRMs capable of cumulative learning. A cumulative learning agent learns and reasons as it interacts with the world. Bayesian networks provide a natural framework for this kind of learning. This is different to those static relational approaches that take a snapshot of the relational database at some time, build the classifier and use it for unseen instances. The dynamic technique proposed in (Pfeffer, 2000) modifies the learned model to represent the reality in a more fair way along the time.

Approaches for mining structural data in form of graph have been proposed too (Holder and Cook, 2000, Gonzalez *et al.*, 2000). In this framework, objects in the data correspond to

vertices in the graph, and relationships between objects correspond to directed or undirected edges in the graph. The Subdue system looks for patterns embedded in graphs. Once a pattern (substructure) is found, is added to the graph in order to simplify it by replacing instances of the substructure with the substructure itself.

3. MULTI-RELATIONAL DATA MINING

This chapter introduces a framework recently proposed to mine relational databases searching through a space consisting of SQL queries only and it is based on the work by Knobbe et al. (Knobbe et al., 1999a, Knobbe et al., 1999b). This is preceded by a brief description of relational notation.

It also describes an implementation of Multi-relational decision tree learning (MRDTL) algorithm for induction of decision trees from relational databases using the previously mentioned approach.

3.1. Relational Vocabulary

A relational database consists of a set of *tables*, $\chi = \{X_1, X_2, ..., X_n\}$ and a set of associations between pairs of tables. These associations can be seen as constraints on how the records from one table relate to records in the other table. Both tables and associations are also known as relations. We will use the distinction between these two terms here.

The columns in a table denote the attributes of that table and the rows correspond to records. Each table can have at least one key attribute that uniquely identifies the records in it; we will denote this attribute as $X.K$. The remaining attributes are either descriptive attributes or foreign key attributes. A foreign key attribute is a key attribute of another table. Foreign keys allow associations between tables to be defined. The nature of this relationship denotes the multiplicity of the association (many-to-one, one-to-many).

Following with the notational convention in (Getoor, 2001), a descriptive attribute A of table X will be denoted $X.A$, and its domain is $D(X.A)$. For a foreign key, the same notation is used. However, for foreign key attributes we need to define their domain type

(Dom[F]) and range type (Range[F]). If $X.F$ is a foreign key attribute in table X and happens to be primary key in table Y, then Dom[F] is X and Range[F] is Y.

Most of associations in a relational database correspond to foreign key relations. These relations can be seen as having two directions. One goes from the table where the attribute is primary key to the table where the attribute is foreign and the other one in the reverse way. Let us call them F and F^{-1}, respectively.

An object in a relational database can consist of several records fragmented across several tables and connected by associations.

As an example, the subset of the data model of the mutagenesis database used in this and the following chapter is described in Fig. 3.1 by mean of an entity-relation diagram.

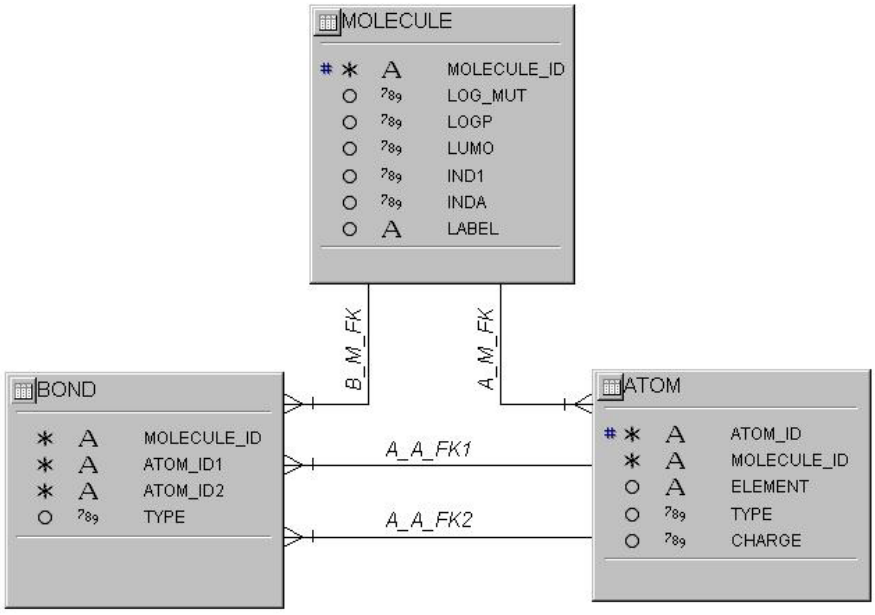


Figure 4.1: Entity-relation diagram for Mutagenesis database.

The figure shows that the database consists of three tables and four relations. A molecule has at least one atom, and at least one bond (this makes a total of at least two atoms per molecule). Table bond describes the type of relation between two atoms; therefore both

atoms must be already in their respective table. The two associations between atom and bond tables represent the latter.

3.2. Multi-Relational Data Mining Framework

In the multi-relational data mining framework (Knobbe *et al.*, 1999a), although the data model can consist of several tables, each describing particular objects' features, only one view of the objects is central to the analysis. The user can choose the kind of objects to analyze by selecting one of the tables as *target table* (this table will be called $T_0 = X_i$ for some $i \in \{1,...,n\}$). The important point here is that *each record in the target table will correspond to a single object in the database*[1]. Once the target table has been selected, a particular *descriptive* attribute of that table can be chosen for classification or regression purposes, this is known as the *target attribute* within the target table.

Given a schema of a relational database and the physical implementation of it, the goal of this approach is to find interesting patterns, that can explain or predict the target attribute within the target table, that not only involve attribute-value descriptions but also structural information denoted by the associations.

Multi-relational patterns can be expressed by using a graphical language consisting of *selection graphs*. These graphs can be translated to SQL or first order logic expressions.

The definition of selection graphs in this work is the one given in (Knobbe *et al.*, 1999b) because it is more adequate to the approach taken here than the other definition in (Knobbe *et al.*, 1999a).

Definition 3.1:

> "A *selection graph* G is a directed graph (N,E), where N is a set of triples (T, C, s) called *selection nodes*, T is a table in the data model and C is a, possibly empty, set of conditions on attributes in T of type $T.A \oplus c$; $\oplus$ is one of the usual selection operators =, <=, >, etc. s is a flag with possible values *open* and *closed*.
>
> E is a set of tuples (p, q, a, e) called *selection edges*, where p and q are selection nodes and a is an association between $p.T$ and $q.T$ in the data model. e is a flag with possible

[1] This point is very important in the discussion of the experimental results in Chapter 4.

values *present* and *absent*. The selection graph contains at least one node n_0 that corresponds to the target table T_0."

Selection graphs can be represented as directed labeled graphs. An example is shown in Fig. 3.2 based on the data model for Mutagenesis database shown in Figure 4.1. The current graph selects those molecules that have at least one atom whose partial charge is less than or equal to -0.392, but for which none of them have charge less than or equal to -0.392 and element equal to 'b' at the same time.

Note, from that figure, that the value of *s* is represented by the presence or absence of a cross in the node, representing the value *open* and *closed* respectively. The value for *e*, in turn, is indicated by the presence (*present* value) or absence (*absent* value) of a cross on the corresponding arrow representing the edge.

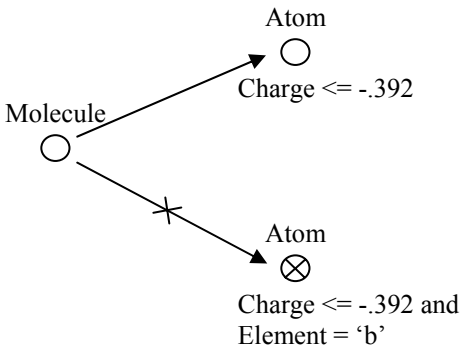


Fig. 3.2: Example of a selection graph for Mutagenesis database.

From the current instance and schema for this relational database, the target table is Molecule, and within molecule the target attribute trying to be predicted is Label although is not shown in the figure.

A *present* edge between tables *p* and *q* represents a join between these two tables using as the join attribute that one that is primary key in *p* and foreign key in *q* with two exceptions to be described later (see adding a negated condition and adding absent edge and closed node).This edge combined with a list of conditions selects those records that match the list of conditions and belong to the join.

On the other hand, an *absent* edge between table p and *q* combined with a list of conditions *q*.C selects those records in *p* that do *not* match the list of condition. Any subgraph that is pointed by an *absent* edge thus corresponds to a set of negative conditions.

Algorithms to translate selection graphs to SQL are given in (Knobbe *et al.*, 1999b). They point out that the resulting SQL statements should not be used in actual implementations. They propose a dedicated architecture (Knobbe *et al.*, 1999a), where these SQL statements are implemented as primitive calls in order to be processed more efficiently. Because the purpose of this work is to test the "goodness" of this approach highlighting strengths and weaknesses, we actually did not implement the dedicated server part. The gathering of statistic is handled by an Oracle DBMS server and the search algorithm run in a different machine, and as we will see later, having a dedicated architecture is the best option for this framework. As a consequence, we directly apply SQL statements to collect statistics in order to find possible refinements for a graph.

Using the algorithms given in (Knobbe *et al.*, 1999b), the graph in Fig. 3.2 translates to the following SQL statement:

```
select distinct T0.mol_id
from Molecule T0, Atom T1
where T0.mol_id = T1.mol_id
and T1.charge <= -0.392
and T0.mol_id not in
        (select T2.mol_id
         from Atom T2
         where T2.charge <= -0.392
         and T2.element = 'b')
```

Basically, the algorithm can be seen as returning, after have traversed a selection graph, a list of the tables (*table_list*) that correspond to the open nodes in the graph, a list of joins (*join*_list) between tables (i.e. conditions such as $T_i.K=T_j.F$, where K is primary key in T_i and foreign key represented by F in T_j), and finally a list of conditions (*condition_list*) on attributes in the appropriate tables. Also, the last list includes the translation of those subgraphs representing negative conditions. Thus, the format of a generic query resulting from a selection graph is the following:

```
select distinct T0.primary_key
from table_list
where join_list
and condition_list
```

Notice that *table_list* can never be empty but *join_list* or/and *condition_list* might be. In these cases, the preceding query should be modified accordingly.

3.2.1. Multi-Relational Decision Tree Learning (MRDTL) Algorithm

Multi-relational decision tree learning algorithm constructs decision tree whose nodes are multirelational patterns i.e., selection graphs.

The algorithm described in (Knobbe *et al.*, 1999b) which we have called MRDTL is an extension of the logical decision tree induction algorithm called TILDE proposed by Blockeel (Blockeel, 1998). TILDE uses first order logic clauses to represent decisions (nodes) in the tree. The data are represented in first order logic rather than a collection of records in a relational database. MRDTL extends TILDE's approach to deal with records in relational databases. They use similar refinements operators, and the way the tree is inducted follows the same logic[2].

Essentially, MRDTL, like the propositional version of the decision tree algorithm (Quinlan, 1987), adds decision nodes to the tree through a process of successive refinement until some termination criterion is met (e.g., correct classification of instances in the training set). Whenever some termination criterion is met, a leaf node with its corresponding class is introduced instead. The choice of decision node to be added at each step is guided by a suitable impurity measure (e.g., information gain) Thus, MRDTL starts with a single node at the root of the tree, which represents the set of all objects of interest in the relational database. This node corresponds to the target table T_0 together with the specified target attribute. The pseudocode for the algorithm is shown in Figure 3.4.

The function *optimal_refinement* considers every possible refinement that can be made to the current pattern G with respect to the database D and selects, in a greedy fashion, the optimal refinement (i.e., one that maximizes information gain). The possible set of refinements to be made at certain point during the process is governed by the current

[2] For short introduction to TILDE refer to (Blockeel and De Raedt, 1997b); for a more detailed one (Blockeel, 1998).

selection graph, the structure of the database D, and the multiplicity of the associations involved. $\overline{G}$ denotes the complement of the selection graph (i.e., it selects objects from the database that are not selected by G.

```
Tree_induction(T: tree, D: database, G: selection graph)
  R := optimal_refinement(G, D)
  if stopping_criteria(G)
        T := leaf(G)
  else
        T_left  := R(G)
        T_right := R_complement(G)
        tree_induction(T_left, D, G)
        tree_induction(T_right, D, G-bar)
        T       := node(T_left, T_right, R)
```

Fig. 3.4: general outline of a decision tree learning algorithm

The induction algorithm outlined before generates binary splits decision trees. A possible modification can be made to consider multiple splits when a condition on some attribute is chosen as the optimal refinement for the graph in question. In this case, we can have one branch for each one of the possible values of the attribute. But, if the optimal refinement to be made is to add an edge and a node then this has only one possible complement.

Another point to highlight about this way of constructing decision trees is the way in which unseen instances will be classified in the future. This is discussed in a later subsection.

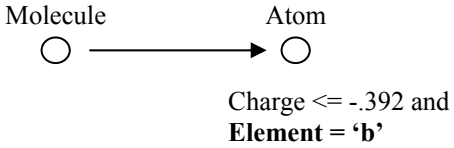
3.2.2. Refinements

As noted above, once the target table and the target attribute have been selected (i.e. the kind of objects central to the analysis have been completely defined) a number of possible *refinements* can be made to the initial node that represents T_0 in order to find a hypothesis to be consistent with the data in the training database.

Knobbe *et al.* (1999a) describe six possible ways of refining selection graphs. Three of them are straightforward refinements for these kinds of graphs. A graph can be specialized by adding a condition, an edge and a node, or by adding an edge between two already existing nodes. But refinements that are designed to deal with multiplicity of the associations in a relational database and the usefulness of look ahead in search are also discussed.

Given the characteristics of the MRDTL algorithm implemented, we are interested on five out of the six refinements previously mentioned. They are: add condition, add edge and node, look ahead, multiple instantiations of associations and mutual exclusion. The refinement that has been left out takes part of an enclosing refinement as it is explained later. All of them are described below using examples from the Mutagenesis database:

- **Mutual exclusion**: algorithms inducing decision trees require that the subsets belonging to patterns derived from the same parent, by applying some kind of refinement, to be mutually exclusive. Because of this, the two main refinements (add a condition and add an edge plus a node) are introduced with their complementary operations. So, the requirement for mutual exclusion introduces two more refinements.
- **Add condition (positive)**: this refinement simply adds a condition to a selection node in *G* without changing its structure. Suppose the graph given below; the refinement to be made was **atom.element = 'b'**. Previous to this operation the set of conditions for atom node contained only the condition atom.charge <= -0.392, and the set of



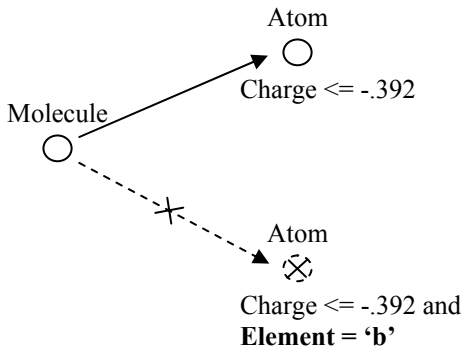
conditions for molecule node was empty. After have added the first condition mentioned, the graph becomes the following:

- **Add condition (negative)**: it is the complementary operation of the previous one. In (Knobbe *et al.*, 1999b) this refinement is explained as follows. If the node that is being refined does not represent the target table, this refinement introduces a new *absent* edge from the parent of that selection node to a new *closed* node that is a copy of the selection node being refined. Its condition list and join list (denoted by the edges coming out from this node) must be copied to the new *closed* node, and the first list must be extended by adding the new condition not negated.

In order to copy the join list (if there is any) of the node being refined to the new *closed* node, it is not necessary to create new nodes similar to those that are related to the first node, it is enough to add copies of its edges from the new node to already existent nodes. Although one of the nodes is the recently added one and the addition of edges to existent nodes is one of the steps within the enclosing refinement discussed here, this point makes necessary the existence of one of the refinements mentioned before: the addition of an edge between two existent nodes.

On the other hand, if the node that is refined represents the target table, the condition is negated and added to the current list of conditions for that node.

Graphically and using the same condition as for the previous example but negated, the resulting selection graph is the following:



In the case that the node that is refined does not correspond to the target table, the simple negated condition can not just be added to the set of conditions of that node for the following reason explained with an example. Let a hypothetical subset of an instance of the Mutagenesis database be that one shown in section a) of Fig. 3.3 for Molecule and Atom tables and the condition to be added is the negation of the aforementioned condition, that is **atom.element != 'b'**.

Then, if just the negated condition is added to the current graph it will correspond to the selection graph and SQL query shown in section b) of the same figure. And the result of that selection is shown in section c). However the correct result should be the empty set. The complementary set of adding the corresponding positive condition is the set of molecules for which none of their atoms have *b* elements.

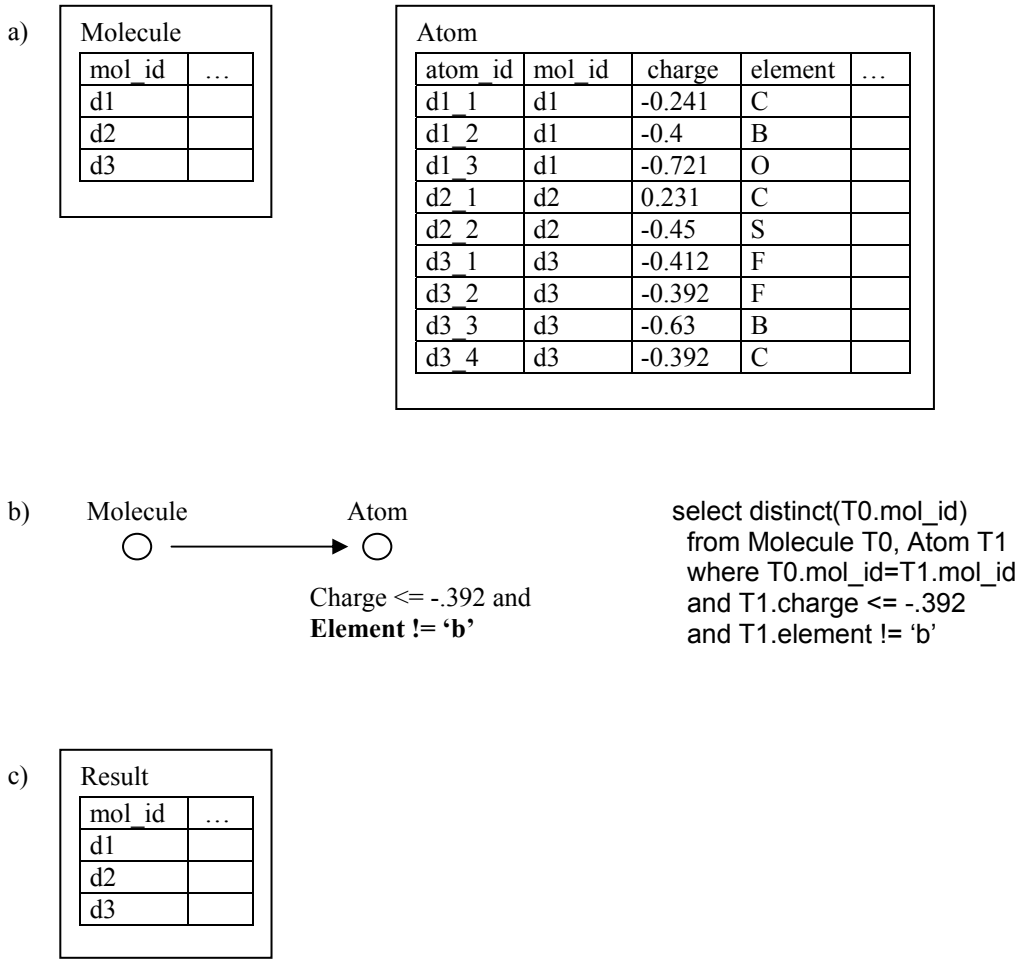


Fig. 3.3: Explanation about adding negative condition.

Therefore the addition of a negative condition to a selection node requires to be handled in the way explained at the beginning. In that case the corresponding SQL query is:

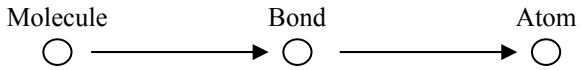
```
select distinct(T0.mol_id)
from Molecule T0, Atom T1
where T0.mol_id=T1.mol_id
and T1.type <= -.392
and T0.mol_id not in
        (select distinct(T1.mol_id)
         from Atom T1
         where T1.charge <= -.392
```

```
and T1.element = 'b')
```

Basically, the inner subquery corresponds to the original graph after have added the positive condition. From the example, the molecules returned by the inner query are *d1*, *d2*, and *d3*. This resulting set must be subtracted from the result of the original graph that is also *d1*, *d2*, and *d3*. Then the final result is the empty set as required.

The first part of the previous implementation of this refinement, when the node to be refined does not correspond to the target table, is only valid for the case where that node is directly connected to the node representing the target table, as it is in the picture drawn before. Otherwise, the whole graph (or a subgraph of it) should be negated.

The following example shows the mentioned in the previous paragraph in a more detailed manner. Suppose the following selection graph at a certain point during the search process:



Let part a) of Fig. 3.4 be a description of the subset of Mutagenesis database that corresponds to the set of records selected by the above graph, and let the condition to be negated and added to the previous graph be **atom.element='n'**.

By using the mechanism for adding negated conditions described in (Knobbe *et al.*, 1999b), the selection graph, after have been refined, will look like that one shown in part b) of Fig. 3.4. Its corresponding translation to SQL is to the right of the same part.

a)

Molecule

mol_id	…
e2	…
e19	…

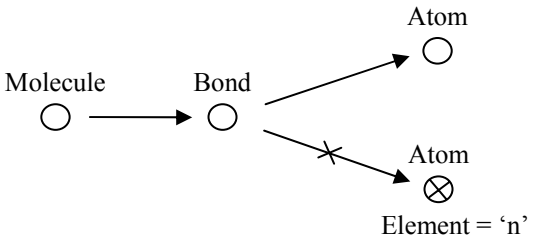
Bond

mol_id	atom_id1	atom_id2
e2	e2_1	e2_2
e2	e2_2	e2_3
e2	e2_3	e2_4
e2	e2_4	e2_5
e2	e2_5	e2_6
e2	e2_6	e2_1
e19	e19_1	e19_2
e19	e19_2	e19_3
e19	e19_3	e19_4

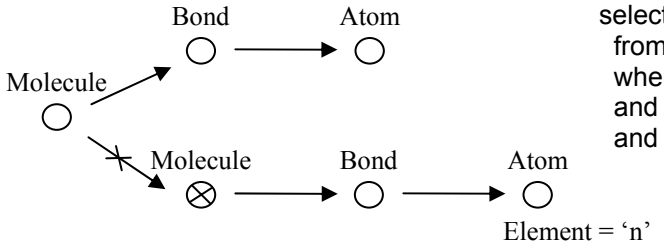
Atom

atom_id	mol_id	element	…
e2_1	e2	C	
e2_2	e2	C	
e2_3	e2	N	
e2_4	e2	N	
e2_5	e2	N	
e2_6	e2	C	
e19_1	e19	C	
e19_2	e19	C	
e19_3	e19	C	
e19_4	e19	C	

b)




```
select distinct(T0.mol_id)
  from Molecule T0, Bond T1, Atom T2
  where T0.mol_id = T1.mol_id
  and T1.atom_id1 = T2.atom_id
  and T1.atom_id1 not in
          (select distinct(T3.atom_id)
            from Atom T3
            where T3.element = ‘n’)
```




```
select distinct(T0.mol_id)
  from Molecule T0, Bond T1, Atom T2
  where T0.mol_id = T1.mol_id
  and T1.mol_id1 = T2.mol_id
  and T0.mol_id not in
          (select distinct(T3.mol_id)
            from Molecule T3, Bond T4,
                  Atom T5
            where T3.mol_id = T4.mol_id
            and T4.atom_id1 = T5.atom_id
            and T5.element = ‘n’)
```

Fig. 3.4: Explanation about adding negative condition when the node object of the refinement is not directly connected to n_0.

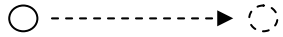
Although the original graph shown prior to Fig. 3.4 selects both molecules, the derivative graphs resulting from the addition of the previous condition and its negation should selects only one molecule each. The graph that includes the positive condition should select *e2* (and it does that). And the graph that includes the negation of that condition should select *e19* (i.e. those molecules for which none of their atoms has element *'n'*). But the graph shown in part b) of Fig. 3.4 corresponding to the first way of adding a negative (complementary) condition selects both molecules: *e2*, and *e19*. Therefore, the correct complementary graph is that one shown in section c).

Thus, for the case where a negated condition has to be added to a node that is not directly connected to the node representing the target table (this does not mean that can not exist a relation in the database between these two tables), it is necessary to complement the whole graph by adding an absent edge from n_0 node to a subgraph representing the original graph plus the condition not negated added to the corresponding node's condition list. The only difference between this new subgraph and the original one is that it has its root node (copy of n_0) closed (see section c of Fig. 3.4 for a graphical example).

The aforementioned process can be made more efficient (in terms of space and time) if instead of copying the complete graph we only copy the subgraph to which the node being refined belongs. That is, we just need to add an absent edge from n_0 to a copy of the first node of the subgraph to which the node where the condition has to be added belongs (Bond node in the example). This copy of the node should be closed. Also the nodes and edges that form the subgraph (Atom and the edge between Bond and Atom in the example) should be copied as they are after the root of the subgraph.

Note that if we follow the first approach (shown in section c) of Fig. 3.4), and not the described in the preceding paragraph, an edge that do not instantiate any association in the data model will connect the target node to a copy of itself but closed. Because of this, that edge should be handled in a different way than those edges that indeed correspond to associations in the relational database.

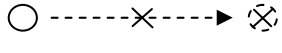
- **Add *present* edge and *open* node**: this refinement instantiates an association in the data model as a *present* edge together with its corresponding table represented as an *open* node and adds these to *G*.



We find it useful to distinguish between two cases of this refinement. Generally, the associations in relational databases are one (from table *p*) to many (in table *q*). In this case, the key attribute in table *p* is a foreign attribute in table *q*. These associations, as said earlier, can be seen as having two directions: *forward* and *backward*. Thus, we can add a present edge from the node representing the table where attribute *F* is primary key to the new open node that represents the table where *F* is foreign key (as proposed in (Knobbe *et al.*, 1999b)). This corresponds to adding an edge and a node in the *forward* direction. However, sometimes it is necessary to add an edge and a node in the *backward* direction. That is, given the current node where *F* is a foreign key, add a present edge to the node representing the node where *F* is primary key. This is especially necessary in learning tasks where the objects to be classified can have multiple class labels for a given assignment of values to their attributes. This would be the case when class labels are not mutually exclusive (Caragea *et al.*, 2002). This point is further elaborated in the following chapter when KDD Cup 2001 (Cheng *et al.*, 2002) datasets are treated.

The modification precedingly outlined also extends MRDTL's classification possibilities by allowing us to consider any attribute within any table in the relational database to be the target attribute.

- **Add *absent* edge and *closed* node**: this refinement is complementary to the previous one and instantiates an association in the data model as an *absent* edge together with its corresponding table represented as a *closed* node and adds these to *G*.

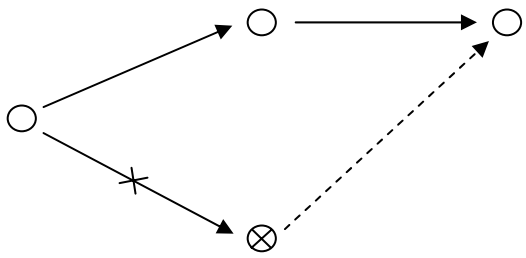


Actually, there are two cases for this kind of refinement too and they follow the same argumentation as described for the previous refinement.

At this point it is worth noting that a *closed* node means that the node in question can not be further refined by adding more conditions on its attributes or edges coming out from it.

For this refinement we need to make the same considerations as when a negated condition is going to be added. Because this refinement should be the complement of the previous one, when the closed node to be added is not directly associated to the target node in the selection graph a procedure similar to that described when a negated condition is added must be followed.

- **Add edge (*present* or *absent*) between two existent nodes**: as explained before, this is not a refinement by its own but it is an important component when adding a negative condition to a node. Graphically, this "refinement" is as follows:



- **Look-ahead refinement**: Sometimes while refining a selection graph, the addition of the optimal refinement may result in no information gain. In general, if a modification to a selection graph does not produce any improvement, then the search through that path is discontinued and a leaf node is introduced instead, even if it may introduce future possible conditions or edges that are relevant to the search process.

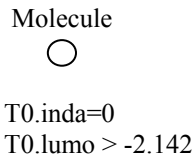
  An example of this scenario occurs when adding an edge from an existing node to a new one it happens that the multiplicity of the association (represented by the edge) between the respective two tables is one to many and each record in the first table has

at least one corresponding record in the other one. The entropy of the new selection graph will be the same as its parent.

For instance, in the Mutagenesis database each molecule has at least one corresponding atom and one corresponding bond associated to it in the respective tables. So, the algorithm previously described will reach a point where no more conditions can be added to the first node because either there are no more conditions to be added or no information gain is obtained with them. Then, one of the possible refinements to be made is to add an edge from Molecule to either Atom or Bond. But, by database definition this will no achieve any improvement because the entropy of the new selection graph does not change respect to its parent due to the number of records to be analyzed is the same as before.

One approach to dealing with this scenario is to provide some sort of look ahead capability to the learner. In the TILDE system (Blockeel, 1998) such a look ahead capability allows several successive refinement steps at once.

Our implementation of MRDTL includes such look ahead capability which is employed in special circumstances when none of the refinements result in a positive information gain although the termination criterion of the algorithm is not met. For instance, suppose one branch of the induced decision tree for the Mutagenesis example has reached the following point:



Furthermore, any other condition that can be added to this node does not improve the classification capability of this graph. We cannot add any edge from this node to a node representing atom or bond table neither since no improvement is achieved. Then, the basic idea is to evaluate the impact of adding a new edge with its corresponding open node plus some condition on an attribute of the new node for one branch and its

negation for the other one. In general, we can consider adding several edges and nodes as part of the look ahead process.

Following the previous example, the refinement with highest information gain is the look ahead shown in Fig. 3.5. The successive refinements performed in this case for the left branch are to add an edge from Molecule to Atom with the corresponding node for Atom table and a condition (charge <= -.392) on table Atom; and for the right branch its negation.

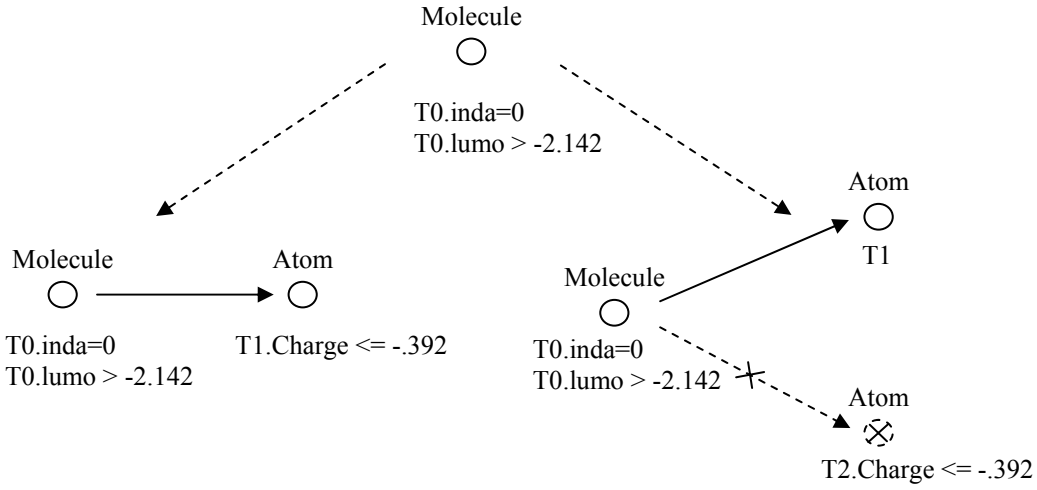


Fig. 3.5: Example of look ahead refinement.

The case shown in Fig. 3.5 (and in general for all look ahead refinements considered here) is different to that of adding a negative condition described before. The node corresponding to the Atom table named *T1* can be removed from the subgraph in the right branch and the remaining graph will still have the same classification power. But, since we are performing several refinements at once it might be the case that later during the search that node can be expanded too.

Look-ahead techniques are computationally expensive but may lead to significant improvements. Because of its computational cost, only two-step look ahead refinements on the possible edges whose respective associations hold the previous constraint are

allowed. But the learner could consider more than one level ahead and the results can be better; it also can consider look-ahead on the conditions to be added to a node.

The TILDE system relies on the user to provide some information in order to determine when look ahead is needed (Blockeel, 1998). Our implementation of MRDTL automatically determines from the relational schema and the current database instance when look ahead might be needed.

- **Multiple instantiation of associations**: actually, this is not a refinement operator. This is the case when multiple instantiations of a particular association exist in a selection graph. This can be possible thanks to the addition of edges in both "directions" of an association or when adding the negation of a condition for a node different to n_0.

3.2.3. Computing Information Gain Associated with a Refinement

Each candidate refinement is evaluated in terms of the possible improvement that can make to the classification accuracy of a selection graph by means of *information gain* (Mitchell, 1997), which involves the characterization of a set of example with respect to the target attribute using *entropy* measure as in the case of the propositional version of the decision tree learning algorithm (Quinlan, 1987). In order to measure information gain for each possible refinement, some kind of statistics' gathering from the database is necessary. For that purpose, a series of queries has been proposed in (Knobbe *et al.*, 1999b), and they are outlined below.

The actual implementation of these queries should be made in a dedicated architecture as primitive calls for efficiency purposes. This is not the case in the current implementation of MRDTL because as said earlier one of the main purposes was to test this new approach to relational data mining in terms of accuracy mainly at this point.

Support of a selection graph

The support of a selection graph (i.e., the number of examples covered by a graph) is computed using the following query:

```
select count(distinct T0.primary_key)
from table_list
where join_list
```

and *condition_list*

The preceding query is known as CountSelection primitive in (Knobbe *et al.*, 1999a). Note that only *join_list* and *condition_list* can be empty, but at least one table will be in *table_list*. If any of the first two lists is empty, then the query should be modified accordingly. This holds for all the following queries.

Evaluating the addition of an edge plus node

For the possible addition of an edge plus a node, we need to calculate the resulting information gain of the new selection graph. The following query produces a histogram that shows the distribution of the number of examples per values of the target attribute. The histogram is used to calculate the entropy for the new multirelational pattern.

select T_0.*target_attribute*, count(distinct T_0.*primary_key*)
from *table_list*
where *join_list*
and *condition_list*
group by T_0.*target_attribute*

The sum of the resulting counts must be equal to the result of the prior query that measures the support of a pattern.

The lists *table_list* and *join_list* correspond to the selection graph being refined plus the addition of the new table (represented by the new node) to the first list and the join condition between a table already in the list and the new "to be added" node to the *join_list*. *Condition_list* remains unchanged.

Evaluating nominal attributes

Each possible pattern resulting from the addition of a condition on some nominal attribute (one pattern for each possible value) of a table is evaluated by means of the following query:

select T_0.*target_attribute*, $T_i.A_j$, count(distinct T_0.*primary_key*)
from *table_list*
where *join_list*
and *condition_list*
group by $T_i.A_j$, T_0.*target_attr*;

The MultiRelationalCrossTable primitive, that is the name given in (Knobbe *et al.*, 1999a), produces a table with the distribution of pairs of values for the target attribute and an arbitrary nominal attribute $T_i.A_j$ in any table T_i. The sum of these counts can exceed the

support of the given pattern if the nominal attribute is not in the target table and multiple records with different values for the selected attribute may correspond to a single record in the target table.

Evaluating numerical attributes

Splits based on numerical attributes are handled using a technique similar to that of C4.5 algorithm (Quinlan, 1993b) with modifications proposed in (Fayyad, 1992, Quinlan, 1996).

The tests considered by C4.5 for continuous attributes are of the kind $A \leq y$, with outcomes true and false. To find the value y of A that maximizes the splitting criterion (information gain), first the cases in the database are sorted on their values of attribute A. Let $v_1, v_2, ..., v_n$ be the list of ordered distinct values for such attribute. Every pair of distinct adjacent values is considered for a potential threshold $y = (v_i + v_{i+1})/2$. The threshold that gives the highest information gain is selected.

Fayyad and Irani in (Fayyad an Irani, 1992) propose and prove that for convex splitting criteria such as information gain it is only necessary to consider pair of distinct values that are adjacent but for which the corresponding instances belong to different classes. This diminishes the number of threshold to be considered.

The other modification proposed by Quinlan in (Quinlan, 1996) is to adjust the information gain of numerical attributes by multiplying it by $\log_2(N\text{-}1)/|D|$, where $N\text{-}1$ is the number of possible thresholds and $|D|$ the number of instances covered by the selection graph. This modification changes the relative bias toward the use of continuous attributes that was outlined as a drawback of C4.5. (Quinlan, 1996).

To obtain the statistics necessary to calculate the information gain resulting from splitting on a continuous attribute the algorithm uses the following query:

```
select target, m, count(*)
from
      (select T0.target_attribute target, T0.primary_key, min(Ti.Aj) m
        from table_list
        where join_list
        and condition_list
        group by T0.target_attribute, T0.primary_key)
group by m, target;
```

Assuming that each record in the target table has multiple associated records in table T_i, for each of these sets the minimum value of the current continuous attribute can be calculated. The occurring minimums will be used as possible splitting thresholds. The fact used here is that testing whether each member of a set has a value for the attribute less than some threshold is equal to test whether the minimum value of that set is less than the threshold. If the assumption at the beginning of this paragraph does not hold (for instance, when the refinement to be made is to add an edge and a node in the "backward" direction or when the numerical attribute to be considered is in the target table), the minimum condition is not necessary and we need to a add one more *group by* clause on $T_i.A_j$.

3.2.4. Classifying Instances

The hypothesis resulting from the relational induction of decision trees described can be viewed as a set of SQL queries associated with the selection graphs that correspond to the leaves of the decision tree. Each selection graph (query) has a class label associated with it. If the corresponding node is not a pure node, (i.e., it does not unambiguously classify the training instances that match the query), the label associated with the node can be based on the classification of the majority of training instances that match the corresponding selection graph. Alternatively, we can use probabilistic assignment of labels based on the distribution of class labels among the training instances that match the corresponding selection graph.

Given a new set of previously unseen instances to be classified, these queries are applied to the database. The set of instances that a query returns will be assigned the class label that the corresponding selection graph has. The complementary nature of the different branches of a decision tree ensures that a given instance will not be assigned conflicting labels.

It is also worth noting that it is not necessary to traverse the whole tree in order to classify a new instance; all the constraints on a certain path are stored in the selection graph associated with the corresponding leaf node. Instances that do not match the selection graphs associated with any of the leaf nodes in the tree are assigned *unknown* label and are counted as incorrectly classified when evaluating the accuracy of the tree on the test data.

3.2.5. Handling Missing Values

The current implementation of MRDTL uses an extra value for each attribute to denote values that may be missing. This extra value is treated in the same way as other possible attribute values. However, this kind of treatment only makes sense when missing values for certain attribute are *representative* of some characteristic. For example, given a database of personnel, for those employees with height greater than certain value that attribute is not measured but left without filling it out (missing value). In this case, those missing values denote those people having height grater than certain threshold.

If we know beforehand the purpose (or possible future purposes) of collecting data, it is possible to minimize or avoid the occurrences of missing values if data mining is one of the possible uses for example. But, most of the data to be mined are results of by-product of some other activity that has the knowledge discovery process as a mean to explain something but not as a goal in itself. In these cases, missing values are likely to be present (Liu *et al.*, 1997) and most of the time they are not representative of any subgroup of instances.

There has been substantial research done on handling missing values in machine learning (Grzymala-Busse and Hu, 2001, Liu *et al.*, 1997, Ortega and Numao, 1999, Quinlan, 1989, Zheng and Low, 1999）. Several of them provide a comparison between different methods of dealing with missing or unknown values (e.g., (Quinlan, 1989)) but others like (Ortega and Numao, 1999) propose new or improved methods. One of the first approaches to tackle this problem was to ignore those instances with missing data (Quinlan, 1987), but soon it was noted that those deleted instances could add useful information to the search process and should be kept and be considered using methods to handle missing values.

As noted before, our implementation of MRDTL supports treatment of missing values as a new attribute value, therefore the straightforward way to cope with missing values that are not representative of any set of instances is to fill them out following some of the techniques proposed for such a purpose. However, we can easily incorporate a variety of more sophisticated approaches for handling missing values that have been proposed in the machine learning literature such as the probabilistic method suggested in (Cestnik *et al.*, 1987) and used for C4.5 (Quinlan, 1993b).

By considering missing values as a new attribute value we assume attributes' independence. That method does not consider dependences between attributes. It is expected that predicting missing values based on the values of other attributes can lead to better accuracy.

In our experiments, we tried with methods based on the most frequent attribute value and the most frequent value for the class of the instance that has the missing value (concept most common attribute value (Grzymala-Busse and Hu, 2001)) as possible candidates for filling in the missing values.

We also could consider the induction of decision trees for each attribute that contains missing values based on the information of other attributes, the original class attribute can be used too (Ortega and Numao, 1999). The attribute values (class labels) are those present in the database, thus those instances having missing data for the class attribute should be deleted. Although this method could be more exact in filling in missing information, it significantly increases the computational cost. This approach is thought to be suitable in domains where a lowering in classification error is worth the increase in computational cost. Other techniques such as Naïve Bayes or Bayesian Networks could be use instead. These approaches seem appropriate for the second data set presented in the next chapter; but their possible application is left for future extensions.

Concept most common attribute value and attribute trees methods mentioned before suffer of a major drawback. As they are described, they can not be used for filling missing values in the test data. Although attribute trees that do not consider the original target attribute to fill in missing information could be an option, the first method mentioned in this paragraph can not be used at all because in real world problems the classes to which the instances to be classified belong are not known. However, in controlled experiments (as it is the case in this work) they can be used, but a comparison with other techniques for handling missing values should be carried out.

The preprocessing of the input data to eliminate missing information is not part of the system yet although all the necessary programs for such a purpose have been done. Their addition should not be a difficult task. Due to this, preprocessing of the data should be done

prior to run the MRDTL algorithm. That is, the instance of relational database to be input should be ready to be mined.

3.3. Description of MRDTL Software

The system implemented can be seen as consisting of three parts. The front-end part, the induction algorithm part which performs the search of relational patterns, and the back-end part.

The front-end of the system is a rudimentary GUI shown in Fig. 3.6 which allows the user to connect to the database to be mined and define in an easy way the numerical and nominal attributes as well as delete those attributes that are not central or of interest to the classification task in question. It also enables the user to specify the target table and target attribute. Finally, an error percentage can be provided by the user. This error percentage can be used to terminate refinement of a path in the decision tree if the error criterion is satisfied by the current node (this serves as a crude pruning method).. The user does not need to type any input (except the error expected), just need to select those attributes that are nominal, numerical, and the target table and target attribute from the corresponding lists shown by the interface. That and the rest of the information used by the algorithm is extracted from the relational schema of the database.

The input to the system can be any Oracle database for which the user wants to find interesting patterns for classification purposes. One of the main constraints that the input database has to hold is that the attributes involved in any of the associations or as primary keys should be single attributes. The software developed does not support composite primary or composite foreign keys yet.

One of the first objectives of this work was to provide to the user with a system where he/she can select any table as the target and within that table any attribute as the classification attribute without worrying to much about the relational representation. This introduces one more constraint on the format of the relational database. That table that is going to be the target must have a primary key and the classification attribute must be nominal (this latter requirement is necessary because the system does not support regression yet).

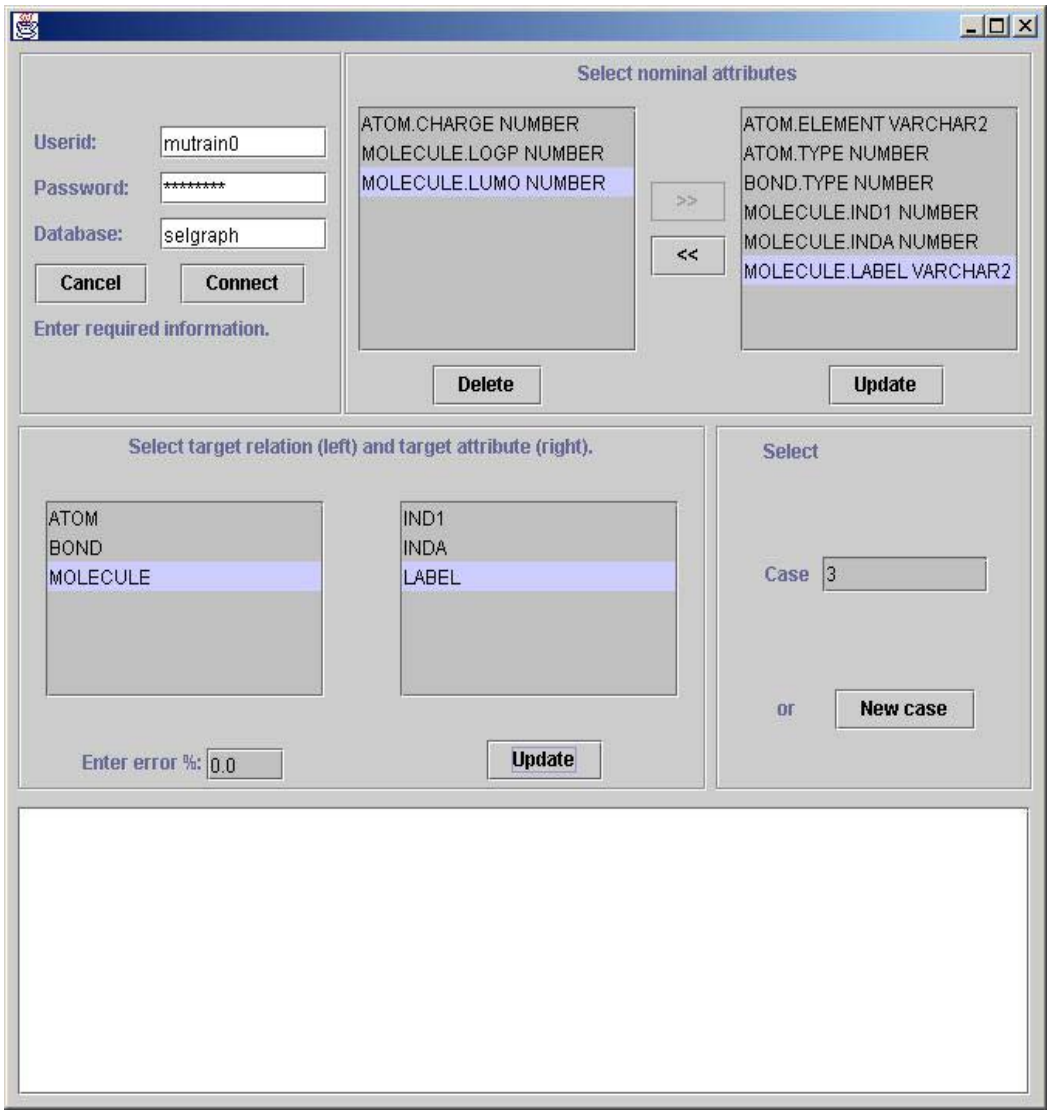


Fig. 3.6: System's Graphic User Interface

The result of learning is the set of SQL queries associated with the selection graphs that correspond to the leaves of the decision tree. That set is enough to classify new instances previously unseen as explained in subsection 3.2.4 and it is shown in the white portion of the GUI in Fig. 3.6. Nevertheless, the complete decision tree is stored into the relational database for future use.

4. EXPERIMENTAL RESULTS

This chapter describes experiments carriede out using MRDTL algorithm on three databases – mutagenesis, gene localization from KDD Cup 2001 and adult database. Results from these experiments are shown, discussed, and compared against other methods.

4.1. Mutagenesis Database

4.1.1. Task Description

This database, available from the Machine Learning Network (MLNet), is one of the most widely used databases in ILP research. For such a reason the original format of this database was Prolog syntax. Therefore, the first step in order to use this database with MRDTL algorithm was to translate it to relational format.

There are several results in the relational data mining literature applying different ILP systems in order to learn from this database. One of our first goals was to compare the performance of our implementation of MRDTL against already existent systems. This is a database that can be used for such a purpose; however, a more thorough comparison is needed.

The entity-relation diagram for the part of the Mutagenesis database used in this study was shown in Chapter 3, figure 3.1 (reproduced below in Fig. 4.1) and briefly described in Section 3.1.

The data set consists of 230 molecules divided into two subsets: 188 molecules for which linear regression yields good results and 42 molecules that are regression-unfriendly. For the regression-friendly molecules, this database instance consists of 4893 atoms and 5243 bonds. For the other set, there are 1001 atoms and 1066 bonds.

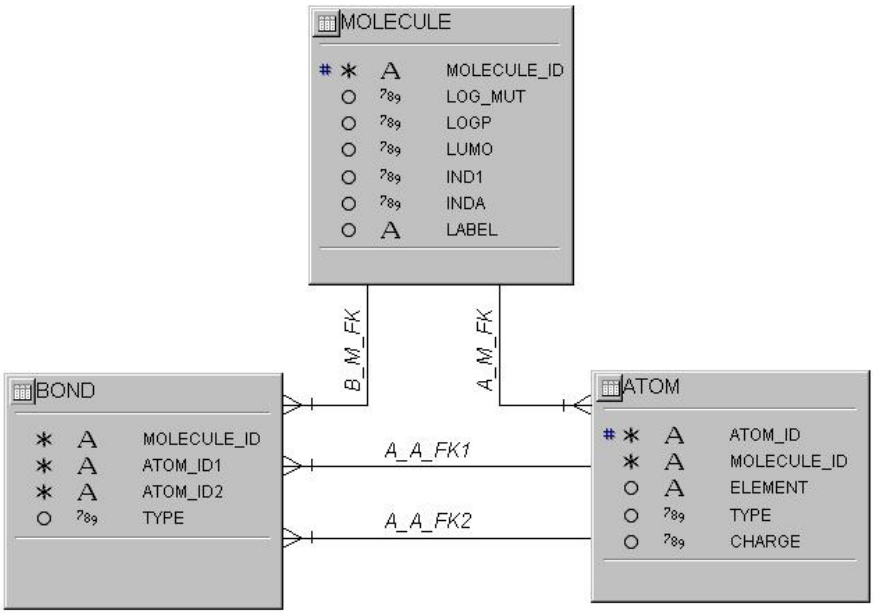


Figure 4.1: Entity-relation diagram for Mutagenesis database.

This database contains descriptions of molecules and the characteristic to be predicted is their mutagenic activity (ability to cause DNA to mutate) represented by attribute *label* in molecule table. This problem comes from the field of organic chemistry and the compounds analyzed are nitroaromatics. These compounds occur in automobile exhaust fumes and sometimes are intermediates in the synthesis of thousands of industrial compounds. High mutagenic level has been found to be carcinogenic.

Originally, mutagenicity has been measured by a real value represented by the attribute *log_mut* in molecule table. But, in most experiments with ILP systems this numerical attribute has been discretized into two values: *active* for those molecules with positive levels of mutagenicity (*log_mut* > 0), and *inactive* for those with zero or negative levels of mutagenicity (*log_mut* <= 0), respectively. Table 4.1 shows the class distribution for the 230 compounds (Srinivasan *et al.*, 1996).

The data model in Fig. 4.1 shows both attributes *log_mut* and *label*. Since MRDTL is limited to learning classifiers, we chose *label* as the target attribute without taking into account *log_mut* during the learning and classification processes.

Table 4.1: Class distribution for Mutagenesis database.

Compounds	Active	Inactive	Total
Regression friendly	125	63	188
Regression unfriendly	13	29	42
Total	138	92	230

A recent study using this database (Srinivasan *et al.*, 1999) recognizes five levels of background knowledge for mutagenesis which can provide richer descriptions of the examples. Table 4.2 shows the five sets of background knowledge[1] where $B_i \subseteq B_{i+1}$ for $i = 0,...,3$. In this study we used only the first three levels of background knowledge in order to compare the performance of MRDTL with other methods for which experimental results are available in the literature.

Table 4.2: Background knowledge for Mutagenesis database extracted from (Srinivasan *et al.*, 1999).

Background	Description
B0	Consists of those data obtained with the molecular modeling package QUANTA. For each compound it obtains the atoms, bonds, bonds types, atom types, and partial charges on atoms.
B1	Consists of Definitions in *B0* plus indicators *ind1*, and *inda* in molecule table.
B2	Variables (attributes) *logp*, and *lumo* are added to definitions in *B1*.
B3	Generic 2-D structures, such as methyl groups, nitro groups, etc., are added to *B2* using atom and bond description.
B4	Using 3-dimensional position of each atom in a molecule, generic 3-D calculations are added to descriptions in *B3*.

4.1.2. Method

We have followed the same methodology that has been used in most of the other studies that have considered this database as a test case. As mentioned earlier, the data can be split into

[1] For a more detailed description of each set of background knowledge the reader should refer to (Srinivasan *et al.*, 1996) and (Srinivasan *et al.*, 1999).

two disjoint subsets. To avoid bias against linear regression (Srinivasan *et al.*, 1996) and to make our results comparable to those in the literature we treat them as two separate subsets.

The classification results obtained from the theory inferred by the learner implemented here are averages over *k*-fold cross-validation; with $k = 10$ for the regression friendly set and $k = 41$ (i.e., leave-one-out cross-validation, because of the relatively small number of samples) for the regression unfriendly data.

4.1.3. Experimental Results

Table 4.2 shows a comparison of the performance of the current implementation of MRDTL with that of Progol (based on results reported in (Blockeel, 1998) and in (Srinivasan *et al.*, 1999)), FOIL (based on results reported in (Blockeel, 1998)), and Tilde (based on results reported in (Blockeel, 1998)).

Due to different hardware has been used for the experiments, the times should be considered to be indicative rather than absolute. Although some ILP systems such as Tilde and FOIL are available from the internet, they are Solaris OS versions only. Therefore, we could not replicate the results in the literature in our systems. The goal of this replication had to be the update of running times only.

The complexity of the theories (as in (Blockeel, 1998)) can not be compared because of the different search spaces the systems have. We can talk about number of literals for ILP systems but that is not applicable for the MRDTL algorithm. But, in the Table 4.4 a comparison of size of decision trees obtained with different levels of background knowledge for MRDTL is shown.

Table 4.3: Accuracy and running time comparisons of Progol, FOIL, Tilde and MRDTL on the set of 188 regression friendly compounds of Mutagenesis database. The numbers represent averages based on ten-fold cross-validation. Entries marked with "--" indicate that the corresponding results are not available at present. Note that the running times are shown mainly to give a general feel for the speed of the algorithms in question. The exact values of the running times are not comparable because of the differences in hardware and software platforms used in the different studies.

Systems	**Accuracy (%)**					**Time (secs.)**				
	B0	*B1*	*B2*	*B3*	*B4*	*B0*	*B1*	*B2*	*B3*	*B4*
Progol[2]	79	86	86	88	89	8695	4627	4974	6530	9587
Progol[3]	76	81	83	88	--	117k	64k	42k	41k	--
Foil[4]	61	61	83	82	--	4950	9138	0.5	0.5	--
Tilde[5]	75	79	85	86	--	41	170	142	352	--
MRDTL	67	87	88	--	--	0.85	332	221	--	--

Results shown in Table 4.3 and Table 4.4 show that the richer description of examples obtained using background knowledge does in fact improve the accuracy of the resulting classifiers. In the case of MRDTL, the improvement obtained by using *B1* as opposed to *B0* is quite pronounced but the improvement resulting from the use of B2 as opposed to *B1* is marginal. It is also clear that different algorithms benefit to varying degrees from the use of different levels of background knowledge. For instance, use of *B1* as opposed to *B0* results in no improvement in FOIL's performance on the regression friendly mutagenesis data (see Table 4.3). This can be explained by the differences in the representational and inductive biases of the algorithms in question.

Table 4.4: Comparison of size of decision trees obtained with the different levels of background knowledge for MRDTL.

Systems	**Number of nodes**		
	B0	*B1*	*B2*
MRDTL	1	53	51

[2] Results for this row were extracted from (Srinivasan *et al.*, 1999).
[3] Results for this row were extracted from (Blockeel, 1998).
[4] Results for FOIL were taken from (Blockeel, 1998).
[5] Results for Tilde were taken from (Blockeel, 1998).

It is interesting to note from Tables 4.3 and 4.4 that MRDTL yields a decision tree with one node when *B0* is used as background knowledge. In this case, in the current implementation of MRDTL, the instances are assigned the class label of the majority class (alternatively, a probabilistic assignment of class labels based on the class distribution at that node could be used). Thus, the classification accuracy (67%) obtained with a single node decision tree can be considered a lower bound on accuracy against which we can compare the different algorithms. Ideally, none of the algorithms should have accuracy lower than 67% on the regression friendly subset of the mutagenesis data. We find that this is in fact true for all of the algorithms studied with the exception of FOIL which has an accuracy of 61% when *B0* and *B1* are used as background knowledge.

Another point worth noting is that MRDTL results in smaller trees when *B2* is used as background knowledge as compared to *B1* and the execution time in the case of *B2* is lower than for *B1*. We can not make a direct comparison of running time but with exception of Progol, the minimum execution time is obtained with *B0*, followed by *B2* and *B3*.

On other hand, MRDTL obtains better results than the other systems in except for *B0*. It is a pending task to try MRDTL with the remaining two sets of background knowledge.

The results from leave one out method for the second set of compounds are shown in Table 4.5 for MRDTL.

Table 4.5: Accuracy, running time, and decision tree size obtained with the MRDTL on the set of 42 regression unfriendly compounds of Mutagenesis database. The numbers represent averages based on 41-fold (i.e., leave-one out) cross-validation.

Background	Accuracy	Time	#Nodes
B0	70 %	0.6 secs.	1
B1	81 %	86 secs.	24
B2	81 %	60 secs	22

Essentially, MRDTL's behavior on the regression unfriendly subset of the Mutagenesis data shows the same general pattern as that in the case of regression-friendly subset. Use of background knowledge *B0* yields a single node decision tree which classified instances according to the label of the majority class. This can be corroborated on the basis of the

distribution of instances shown in Table 4.1: a majority of 70% of the regression-unfriendly instances are inactive compounds.

There is not improvement in accuracy resulting from the use of *B2* over *B1* but the running time using *B2* is lower than that for *B1*. Although an accuracy of 81 % was achieved, the experiments carried out using Progol in (Srinivasan *et al.*, 1996) show an average accuracy of 83 %.

Two recent approaches, (Sebag and Rouveirol, 1997) and (Kramer and De Raedt, 2001) have reported a maximum accuracy of 93.6% and 94.7%, respectively for mutagenesis database. The approach taken by Sebag and Rouveirol (1997) comes from the field of ILP and is concerned with polynomial induction and use of first order logic hypotheses with no size restriction. Instead of exhaustively exploring the set of matchings between any example and any candidate hypothesis, the user determines the number of matchings samples to consider; thus, controlling the cost of induction and classification. The experiments do not specify if the whole data set was used or two disjoint sets were considered; they did not use 10-fold cross validation, but the data set was randomly divided into a training set including 90% of the data and a test set with the remainder. The result was averaged on 25 independent selections of these sets. Kramer and De Raedt (2001) propose a novel feature construction method where the user can specify the features of interest using a conjunctive query. The solutions to the query are organized in a version space and the resulting features can be used by traditional attribute-value machine learning algorithms. Particularly, they used the propositional learners C4.5, PART (Frank and Witten, 19980), logistic regression and linear support vector machines available in the Weka workbench (Witten and Frank, 1999). In their experiments, they only used 2-D information (subset *B3* of background knowledge). They did not use partial charges, atom types, functional groups or the like, but LUMO and logP values were considered. The experiments description do not specify if they considered the two disjoint sets of compounds as described above or just the complete data set.

4.2. Gene Localization Database

This database was used in the last competition on data mining hold in conjunction with the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2001) and it is available online (Cheng *et al.*, 2002).

KDD Cup 2001 was focused on data from genomic and drug design applications and involved three tasks based on two data sets. Here, we focus on only one data set and one task. The election of the data set was based on its relational nature.

4.2.1. Database and Task Description

The three tasks that KDD Cup 2001 involved were: prediction of molecular bioactivity for drug design, prediction of gene/protein function and localization. Although both last tasks may involve a relational approach, in the case of gene/protein function prediction, an instance may have several class labels. MRDTL, much like its propositional counterpart C4.5, assumes that each instance can be assigned to only one of several non-overlapping classes. Thus we, in MRDTL's current form, decided to focus the experiments on the gene/protein localization task where each instance has a single class label. We will use the name *Gene Localization database* to refer to the data for the gene/protein localization task.

Two variants of the data set for the mentioned task were provided. The first version consists of a single table with 2960 attributes and it is appropriate for propositional learning. The second consists of two tables with 13 attributes in total. We used the second variant due to its relational nature but not as it was given because those two tables were not in normal form as it is required by the relational algorithm.

The names of the two original tables are *genes_relation* and *interactions_relation*. The *genes_relation* table contains 862 different genes but there could be more than one row in the table for each gene. The attribute *gene_id* identifies a gene uniquely. Since our current implementation of MRDTL requires that the target table must have a primary key, it was necessary to normalize *gene_relation* table before we can use it as the target table. This normalization was achieved by creating the tables named *gene, interaction, and composition* as follows: attributes in the *genes_relation* table that did not have unique values for each gene were placed in the *composition* table and the rest of the attributes were placed in the

gene table. The *gene_id* attribute is primary key in the *gene* table and foreign key in the *composition* table. The *interaction* table is identical to the original *interactions_relation* table. This represents one of several ways of normalizing the original tables. This renormalization of the relational database given is different from the one described in (Cheng *et al.*, 2002). It is possible that the particular renormalization used might have an impact on the performance of different algorithms. It is left for future experiments to compare the performance of the algorithm using both renormalizations.

The entity-relation diagram for the normalized version of the *Gene Localization database* is shown in Figure 4.2. For this task, the target table is *gene* and the target attribute is *localization*. From this point of view, the training set consists of 862 genes and the test set 381. If we want to predict *function*, the number of instances in the training set will be 4346, that is, the number of distinct records in *composition* table. In this latter case, it might be necessary to follow an association in the backward direction as explained in Chapter 3 if the algorithm has to consider the remaining tables and attributes.

The experiments described here focused on building a classifier for predicting the localization of proteins by assigning the corresponding instance to one of 15 possible localizations.

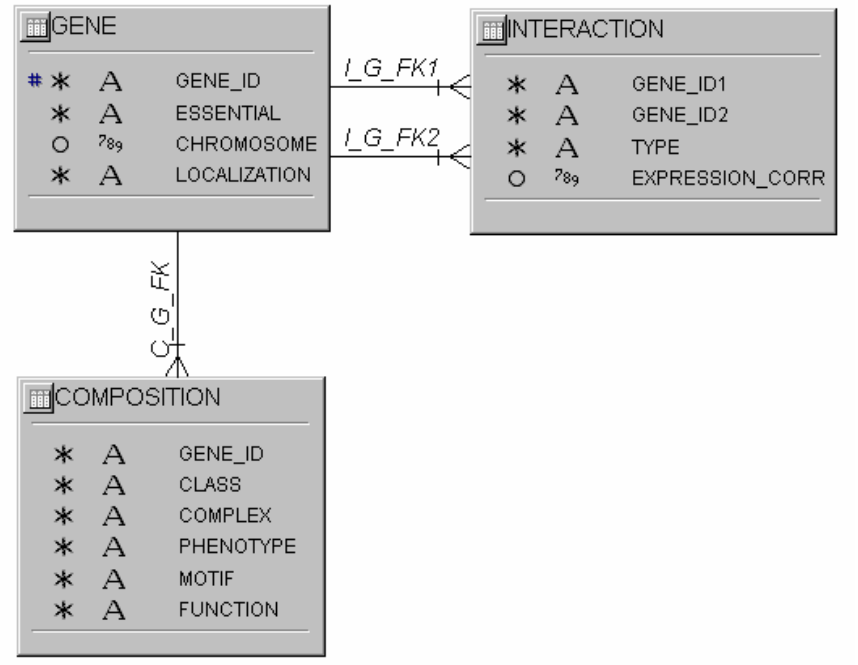


Figure 4.2: entity-relation diagram for *Genes Localization* data set

4.2.2. Method

For this task, we followed the general procedure used in the KDD Cup competition namely, construct a classifier using all of the training data and test the resulting classifier on the test set provided (Cheng *et al.*, 2002).

4.2.3. Discussion of Results

Gene localization task presents significant challenges because many attribute values in the training instances corresponding to the 862 training genes are missing. Initial experiments using a special value (*missing*) to encode a missing value for an attribute resulted in classifiers whose accuracy is around 50 % on the test data that also have many missing attribute values.

This prompted us to investigate incorporation of more sophisticated approaches to handling missing values that have been proposed in the literature. As noted in the previous chapter we use techniques for filling in the missing values. The methods used were concept most common attribute value, majority value and eventually we will use decision trees for all

the attributes with missing values or other probabilistic methods that consider information available from other attributes such as Naïve Bayes.

Replacing missing values by the most common value of the attribute for the class during training resulted in an accuracy of around 85% with a decision tree of 367 nodes. This result was achieved by allowing the traversing of associations in the *forward* and *backward* direction and not placing any limit in the number of times an association can be instantiated. If we heuristically limit the number of times an association can be instantiated but still using the same method to fill in missing values, the accuracy lowered to 80%. Finally, if we allow the algorithm to follow associations only in the forward direction, then an accuracy of around 75% is obtained. This shows that providing reasonable guesses for missing values can significantly enhance the performance of MRDTL on real world data sets. However, in practice, since the class labels for test data are unknown, it is not possible to replace a missing attribute value by the most frequent value for the class during testing, so the use of alternative methods of handling missing values are mandatory.

Based on these preliminary results, we conclude that there is a need for incorporating more sophisticated techniques for handling missing values (e.g., predicting missing values based on the values of other attributes). Work in progress is aimed at extending the current implementation of MRDTL to include principled approaches for dealing with missing values that can be applied in a realistic setting.

4.3. Adult Database

4.3.1. Database and Task Description

This is the simplest of the three databases used. It consists of only one table and it is suitable for propositional learning with 6 numerical attributes and 8 nominal attributes. It is available online at (UCI, Machine Learning Repository).

The information in this database was extracted from the 1994 census database and the prediction task is to determine whether a person makes over $50,000 a year (i.e., attribute being predicted is salary greater or less than 50,000). For this purpose, the class attribute values have been discretized into two values: >50k and <=50k.

The purpose of using this database was to test the implemented MRDTL algorithm in a purely propositional setting. There are some results in the literature using this database (e.g. (Kohavi, 1996)) against which we can compare our results.

Table 4.6: Class distribution for Adult database.

	Training		**Test**		**Total**
	>50k	<=50k	>50k	<=50k	
w/ missing values	7841	24720	3846	12435	48842
w/o missing values	7508	22654	3700	11360	45222

4.3.2. Method and Results

Most of the results in the literature that used this database were obtained by following the same methodology: removal of unknowns and use of the original train/test split.

Adult database domain is hard with a considerable number of records. Removal of instances with missing values resulted in an accuracy of 82.2% on the original test set. This can be compared to the accuracy reported in (Kohavi, 1996) of 84.46+-0.30 for C4.5, 83.88+-30 for Naïve-Bayes and 85.90+-0.28 for NBTree (a decision tree Naïve-Bayes hybrid approach). The current MRDTL version should be augmented with pruning and other characteristics in order to make these results really comparable to those of C4.5 in a purely propositional setting. But the preliminary results reported are promising and deeper research has to be done in order to make the current version more efficient and more competitive.

5. SUMMARY and DISCUSSION

Learning from relational databases and from data describing complex/structured objects is an important problem in machine learning nowadays. Most of real-world data are organized in relational format, with databases consisting of multiple relations and not just one as typical data mining approaches assume. Also, structured types of data such as in bioinformatics, computational biology, HTML and XML documents require the use of a more expressive pattern language than propositional.

In this context, (multi)-relational data mining deals with knowledge discovery from database consisting of multiple tables directly, without the need to squeeze data fragmented across several tables into a single one.

(Multi)-relational data mining approaches do not only focus on relational database but also deductive databases (e.g., ILP systems). They consider all of the main data mining tasks (i.e. association analysis, classification, clustering, learning probabilistic models and regression) and several of them are extensions of their single-table counterparts to the multiple-table case.

In this paper, we have described an implementation of multi-relational decision tree learning (MRDTL) algorithm based on the techniques proposed by Knobbe *et al*. (Knobbe *et al*., 1999a, Knobbe *et al*., 1999b).

The main contribution of this paper is the empirical evaluation of the MRDTL system, for which there are no results on the literature, on widely used data sets for relational learning such as mutagenesis database, databases from the last KDD competition, and one propositional database. We also gained insights about the algorithm and they resulted in the proposal of two important modifications described in Chapter 3 and future extensions in term of storage and speed efficiency that are currently being explored.

Results of our experiments on the widely used Mutagenesis database indicate that MRDTL offers a promising alternative to existing algorithms such as Progol (Muggleton,

1995), FOIL (Quinlan, 1993a), and Tilde (Blockeel, 1998). Preliminary results of our experiments with the protein/gene localization task (based on the data from the KDD Cup 2001 competition) indicate that MRDTL, if equipped with principled approaches to handling missing attribute values, can be an effective algorithm for learning from relational databases in real-world applications. Finally, results of experiments on a pure propositional database show that this algorithm is also an alternative for this kind of learning.

Work in progress and for the future is aimed at:

- Incorporation of sophisticated methods for handling missing attribute values into MRDTL
- Incorporation of sophisticated pruning methods or complexity regularization techniques into MRDTL to minimize overfitting and improve generalization
- More extensive experimental evaluation of MRDTL on real-world data sets
- Development of ontology-guided multi-relational decision tree learning algorithms to generate classifiers at multiple levels of abstraction, based on the recently developed propositional decision tree counterparts of such algorithms (Zhang *et al*., 2002)
- Development of variants of MRDTL for classification tasks where the classes are not disjoint, based on the recently developed propositional decision tree counterparts of such algorithms (Caragea *et al.*, 2002)
- Development of variants of MRDTL that can learn from heterogeneous, distributed, autonomous data sources based on recently developed techniques for distributed learning (Caragea *et al.*, 2001a; Caragea *et al.*, 2001b) and ontology-based data integration (Honavar *et al*., 2001; Honavar *et al.*, 2002; Reinoso-Castillo, 2002).
- Application of multi-relational data mining algorithms to data-driven knowledge discovery problems in bioinformatics and computational biology.
- One of the most important drawbacks we identified using this framework for relational induction of decision trees is that sometimes extremely complex or long queries that correspond to the leaves of the tree result. Alternative techniques to deal with this problem and to make the algorithm more efficient are being explored.

- In a more specific context, it is of interest to try MRDTL algorithm with the two remaining sets of background knowledge for mutagenesis database.
- In the context of first-order extensions of probabilistic models (Bayesian Networks), the method proposed in (Cheng et al., 2002) based on the use of *Markov blanket* concept together with information gain based feature filtering can be combined with first order extensions of BNs to obtain more compact models but with the same predictive power.

REFERENCES

[Agrawal *et al.*, 1996] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H, and Verkamo, A. I. Fast discovery of association rules. In U. Fayyad, G. Piatesky-Shapiro, R. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, MIT Press, 1996.

[Blockeel and De Raedt, 1997a] Blockeel, H., and De Raedt, L. Relational Knowledge Discovery in Databases. In *Proceedings of the 6th International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 1997.

[Blockeel and De Raedt, 1997b] Blockeel, H., and De Raedt, L. Top-down induction of Logical Decision Trees. In W. K. Van Marcke, editor, *Proceedings of the Ninth Dutch Conference on Artificial Intelligence (NAIC'97)*, 1997.

[Blockeel, 1998] Blockeel, H. Top-down induction of first order logical decision trees. PhD dissertation, Department of Computer Science, Katholieke Universiteit Leuven, 1998.

[Caragea *et al.*, 2001a] Caragea, D., Silvescu, A., and Honavar, V. Invited Chapter. Towards a Theoretical Framework for Analysis and Synthesis of Agents That Learn from Distributed Dynamic Data Sources. In *Emerging Neural Architectures Based on Neuroscience*, Springer-Verlag, 2001.

[Caragea *et al.*, 2001b] Caragea, D., Silvescu, A., and Honavar, V. Learning Classification Trees from Distributed Data. Technical Report TR-2001-10, Department of Computer Science, Iowa State University, Ames, Iowa 50011-1040, 2001.

[Caragea *et al.*, 2002] Caragea, D., Silvescu, A., and Honavar, V. Learning decision tree classifiers when the classes are not disjoint. In preparation, 2002.

[Cestnik *et al.*, 1987] Cestnik, B., Kononenko, I., and Bratko, I. Assistant-86: A knowledge-elicitation tool for sophisticated users. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning*, Sigma Press, 1987.

[Cheng *et al.*, 2002] Cheng, J., Krogel, M., Sese, J., Hatzis, C., Morishita, S, Hayashi, H., and Page, D. KDD Cup 2001 Report. *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) Explorations*, volume 3, issue 2, January 2002. http://www.acm.org/sigs/sigkdd/explorations/ (Last date accessed: July 9th, 2002)

[Cussens, 1999] Cussens, J. Loglinear models for first-order probabilistic reasoning. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, Kaufmann, 1999.

[Dehaspe and De Raedt, 1997] Dehaspe, L., and De Raedt, L. Mining association rules in multiple relations. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297of *Lecture Notes in Artificial* Intelligence, Springer-Verlag, 1997.

[De Raedt, 1997] De Raedt, L. Logical settings for concept-learning. *Artificial Intelligence*, volume 95, issue 1, 1997.

[De Raedt, 1998] De Raedt, L. Attribute-value learning versus Inductive Logic Programming: the Missing Links (Extended Abstract). In *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 1998.

[De Raedt *et al.*, 2001] De Raedt, L., Blockeel, H., Dehaspe, L., and Van Laer, W. Three companions for data mining in first order logic. In (Dzeroski and Lavrac, 2001).

[Dietterich *et al.*, 1997] Dietterich, T. G., Lathrop, R. H., and Lozano-Pérez, T. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, volume 89, 1997.

[Dzeroski *et al.*, 2001] Dzeroski, S., De Raedt, L., and Driessens, K. Relational Reinforcement Learning. Report CW 311, Department of Computer Science, Katholieke Universiteit Leuven, 2001.

[Dzeroski and Lavrac, 2001] Dzeroski, S. and Lavrac, N. editors. Relational Data Mining. Springer-Verlag, 2001.

[Fayyad an Irani, 1992] Fayyad, U. M. and Irani, K. B. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, volume 8, 1992.

[Frank and Witten, 1998] Frank, E. and Witten, I. H. Generating Accurate Rule Sets without Global Optimization. In *Proceedings of the 15th International Conference on Machine Learning (ICML-98)*, Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[Friedman *et al.*, 1999] Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, Morgan Kaufman, 1999.

[Friedman *et al.*, 2001] Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. Learning probabilistic relational models. In (Dzeroski and Lavrac, 2001).

[Friedman and Koller, 2001] Friedman, N., and Koller, D. Tutorial: Learning Bayesian Networks from Data. *Neural Information Processing Systems: Natural and Synthetic (NIPS 2001)*, 2001.
http://www-2.cs.cmu.edu/Web/Groups/NIPS/NIPS2001/nips2001.html
(Last date accessed: July 9th, 2002)

[Getoor, 2001] Getoor, L. Multi-relational data mining using probabilistic relational models: research summary. In (Knobbe and Van der Wallen, 2001).

[Gonzalez *et al*., 2000] Gonzalez, J., Jonyer, I., Holder, L. B., and Cook, D. J. Efficient Mining of Graph-Based Data. In *Proceedings of AAAI 2000 Workshop on Learning Statististical Models from Relational Data*, AAAI Press, 2000.

[Grzymala-Busse and Hu, 2001] Grzymala-Busse, J. W., and Hu, M. A comparison of several approaches to missing attribute values in data mining. In *Proceedings of the Second International Conference on Rough Sets and Current Trends in Computing*, *RSCTC* 2000, volume 2005 of *Lecture Notes in Computer Science*, Springer, 2001.

[Holder and Cook, 2000] Holder, L. B., and Cook, D. J. Graph-based data mining. *IEEE Intelligent Systems* 15, 2000.

[Information Discovery] Information Discovery Inc. OLAP and DataMining; Bridging the Gap. http://www.datamining.com/bridge.htm (Last date accessed: July 9th, 2002)

[Jaeger, 1997] Jaeger, M. Relational Bayesian networks. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-1997)*, 1997.

[Jensen, 2001] Jensen, F. V. Bayesian Networks and Decision Graphs. Springer-Verlag, 2001.

[Karalic and Bratko, 1997] Karalic, A. and Bratko, I. First order regression. *Machine Learning*, volume 26, 1997.

[Kersting and De Raedt, 2000] Kersting, K., and De Raedt, L. Bayesian Logic Programs. In *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, 2000.

[Kersting *et al.*, 2000] Kersting, K., De Raedt, L., and Kramer, S. Interpreting Bayesian Logic Programs. In L. Getoor and D. Jensen, editors, *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, AAAI Press, 2000.

[Kietz and Wrobel, 1992] Kietz, J-U. and Wrobel, S. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive logic programming*, Academic Press, 1992.

[Knobbe *et al.*, 1999a] Knobbe, J., Blockeel, H., Siebes, A., and Van der Wallen, D. M. G. Multi-relational Data Mining. In *Proceedings of Benelearn '99*, 1999.

[Knobbe et al, 1999b] Knobbe, J., Siebes, A., and Van der Wallen, D. M. G. Multi-relational decision tree induction. In *Proceedings of the 3rdEuropean Conference on Principles and Practice of Knowledge Discovery in Databases,* PKDD '99, 1999.

[Knobbe *et al.*, 2001] Knobbe, A. J., Siebes, A., Blockeel, H., and Van der Wallen, D. M. G. Multi-relational data mining using UML for ILP. In (Knobbe and Van der Wallen, 2001).

[Knobbe and Van der Wallen, 2001] Knobbe, A. J. and Van der Wallen, D. M. G. editors. *Proceedings of the First Workshop in Multi-relational Data Mining*, 2001.

[Kohavi, 1996] Kohavi, R. Scaling up the accuracy of Naïve Bayes Classifiers: a decision tree hybrid. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[Koller, 1999] Koller, D. Probabilistic Relational Models. In S. Dzeroski and P. Flach, editors, *Proceedings of 9th International Workshop on Inductive Logic Programming (ILP-99)*, Springer, 1999.

[Kramer and De Raedt, 2001] Kramer, S. and De Raedt, L. Feature Construction with Version Spaces for Biochemical Applications. In *Proceedings of the 18th International Conference on Machine Learning (ICML-2001)*, 2001.

[Kramer, 1996] Kramer, S. Structural regression trees. In *Proceedings of the 13th National Conference on Artificial Intelligence*, 1996.

[Krogel and Wrobel, 2001] Krogel, M. and Wrobel, S. Transformation-Based Learning Using Multirelational Aggregation. In Céline Rouveirol and Michèle Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2001.

[Lari and Young, 1990] Lari, K. and Young, S. J. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, volume 4, 1990.

[Lavrac *et al.*, 1991] Lavrac, N., Dzeroski, S., and Grobelnik, M. Learning nonrecursive definitions of relations with LINUS. In *Proceedings of the 5th European Working Session on Learning*, Springer, 1991.

[Lavrac and Dzeroski, 1994] Lavrac, N. and Dzeroski, S. Inductive Logic Programming: Techniques and Applications. Ellis Horwood, 1994.

[Lavrac, 2002] Lavrac, N. Introduction to Inductive Logic Programming. Computer Science course ComS70301: *Learning from Structured Data*, University of Bristol, Department of Computer Science.
http://www.cs.bris.ac.uk/Teaching/Resources/COMS70301/
(Last date accessed: July 9th, 2002)

[Lindner and Morik, 1995] Lindner, G., and Morik, K. Coupling a relational learning algorithm with a database system. In Y. Kodratoff, G. Nakhaeizadeh, and C. Taylor, editor, *Workshop Notes of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, 1995.

[Liu *et al.*, 1997] Liu, W. Z., White, A. P., White, S. G., Thompson, S. G., and Bramer, M. A. Techniques for dealing with missing values in classifications. In X. Liu, P. Cohen and M. Berthold, editors, *Advances on Intelligent Data Analysis*, Springer-Verlag, 1997.

[MLNet] The Machine Learning Network, Online Information Service.
http://www.mlnet.org/ (Last date accessed: July 9th, 2002)

[Mitchell, 1997] Mitchell, T. Machine Learning. McGraw Hill, 1997.

[Muggleton, 1995] Muggleton, S. H. Inverse entailment and progol. *New Generation Computing*, 1995.

[Muggleton, 1996] Muggleton, S. H. Stochastic Logic Programs. In *Proceedings of the 5th International Workshop on Inductive Logic Programming*, 1996.

[Muggleton, 2000] Muggleton, S. H. Learning Stochastic Logic Programs. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, AAAI Press, 2000.

[Neville and Jensen, 2000] Neville, J., and Jensen, D. Iterative Classification in Relational Data. AAAI 2000 Workshop on Learning Statistical Models from Relational Data, 2000. http://robotics.stanford.edu/srl/workshop.html (Last date accessed: July 9th, 2002)

[Ngo and Haddawy, 1997] Ngo, L., and Haddawy, P. Answering queries form context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 1997.

[Ortega and Numao, 1999] Ortega, O., and Numao, M. Ordered estimation of missing values. In *Proceedings of the 3rd Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-99)*, Springer-Verlag, 1999.

[Pearl, 1991] Pearl, J. Reasoning in Intelligent Systems: Networks of Plausible Inference. *Morgan Kaufmann*, 2nd edition, 1991.

[Pfeffer, 2000] Pfeffer, A. A Bayesian Language for Cumulative Learning. In *Proceedings of AAAI 2000 Workshop on Learning Statististical Models from Relational Data*, AAAI Press, 2000.

[Quinlan, 1987] Quinlan, J. R. Induction of decision trees. *Machine Learning*, volume 1, 1987.

[Quinlan, 1989] Quinlan, J. R. Unknown attribute values in induction. In *Proceedings of the 6th International Machine Learning Workshop*, 1989.

[Quinlan, 1993a] Quinlan, J. R. FOIL: A midterm report. In *Proceedings of the 6th European Conference on Machine Learning,* volume 667 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 1993.

[Quinlan, 1993b] Quinlan, J. R. C4.5: Programs for Machine Learning. San Mateo: Morgan Kaufmann, 1993.

[Quinlan, 1996] Quinlan, J. R. Improved Use of Continuous Attributes in C4.5. *Journal of Artificial Intelligence Research*, AI Access Foundation and Morgan Kaufmann Publishers, volume 4, 1996.

[Quinlan, Rulequest] Quinlan, J. R. RuleQuest Research Pty Ltd. http://www.rulequest.com/ (Last date accessed: July 9th, 2002)

[Reinoso-Castillo, 2002] Reinoso-Castillo, J. Ontology-Driven Query-Centric Federated Solution for Information Extraction and Integration from Autonomous, Heterogeneous, Distributed Data Sources. Masters Thesis, Department of Computer Science, Iowa State University, 2002.

[Sebag and Rouveirol, 1997] Sebag, M. and Rouveirol, C. Tractable Induction and Classification in First Order Logic via Stochastic Matching. In *Proceeding of the 15th International Joint Conference on Artificial Intelligence*, 1997.

[Srinivasan *et al.*, 1996] Srinivasan, A., Muggleton, S., Sternberg, M., and King, R. D. Theories for mutagenicity: a study in first-order and feature-based induction. *Artificial Intelligence*, volume 85, 1996.

[Srinivasan *et al.*, 1999] Srinivasan, A., King, R. D., and Muggleton, S. The role of background knowledge: using a problem from chemistry to examine the performance of an ILP program. Technical Report PRG-TR-08-99, Oxford University Computing Laboratory, Oxford, 1999.

[UCI, Machine Learning Repository] University of California, Irvine Machine Learning Repository. http://www.ics.uci.edu/~mlearn/MLRepository.html (Last date accessed: July 9th, 2002)

[Witten and Frank, 1999] Witten, I. H. and Frank, E. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann, 1999.

[Zhang *et al.*, 2002] Zhang, J., Silvescu, A., and Honavar, V. Ontology-Driven Induction of Decision Trees at Multiple Levels of Abstraction. In *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation (SARA-2002)*, 2002.

[Zheng and Low, 1999] Zheng, Z., and Low, B.T. Classifying unseen cases with many missing values. In *Proceedings of the 3rd Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-99)*, Springer-Verlag, 1999.