, , 1–29 ()

A Multistrategy Approach to Relational Knowledge Discovery in Databases

KATHARINA MORIK AND PETER BROCKHAUSEN

{morik, brockh}@ls8.informatik.uni-dortmund.de

Univ. Dortmund, Computer Science Department, LS VIII, D–44221 Dortmund

Received May 1, 1991

Editor:

Abstract. When learning from very large databases, the reduction of complexity is extremely important. Two extremes of making knowledge discovery in databases (KDD) feasible have been put forward. One extreme is to choose a very simple hypothesis language, thereby being capable of very fast learning on real-world databases. The opposite extreme is to select a small data set, thereby being able to learn very expressive (first-order logic) hypotheses. A multistrategy approach allows one to include most of these advantages and exclude most of the disadvantages. Simpler learning algorithms detect hierarchies which are used to structure the hypothesis space for a more complex learning algorithm. The better structured the hypothesis space is, the better learning can prune away uninteresting or losing hypotheses and the faster it becomes.

We have combined inductive logic programming (ILP) directly with a relational database management system. The ILP algorithm is controlled in a model-driven way by the user and in a data-driven way by structures that are induced by three simple learning algorithms.

Keywords: Knowledge discovery in databases, inductive logic programming, functional dependencies, numerical intervals, background knowledge

1. Introduction

Knowledge discovery in databases (KDD) is an application that challenges machine learning because it has both high efficiency requirements and also high understandability and reliability requirements. Learning is complicated because the learning task is to find all interesting, valid and non–redundant rules (rule learning). This learning task is more complex than the concept learning task, as was shown by Uwe Kietz (Kietz, 1996). To make it even worse, the data sets for learning are very large.

Two extremes of making KDD feasible have been put forward. One extreme is to choose a very simple hypothesis language which allows one to do very fast learning on real-world databases. Fast algorithms have been developed that generalize attribute values and find dependencies between attributes. These algorithms are capable of directly accessing a database, i.e. the representation language $\mathcal{L}_\mathcal{E}$ is the language of the database. The APRIORI and APRIORITID algorithms find *association rules* that determine subsets of correlated attribute values (Agrawal et al., 1996). Attribute values are represented by turning each attribute value into a Boolean

Table 1. Relational database with two tables: potential_customer and married_to

potential_customer:

person	age	sex	income	customer
Ann Smith	32	f	10,000	yes
Joan Gray	53	f	1,000,000	yes
Mary Blythe	27	f	20,000	no
Jane Brown	55	f	20,000	yes
Bob Smith	30	m	100,000	yes
Jack Brown	50	m	200,000	yes

married_to:

husband	wife
Jack Brown	Jane Brown
Bob Smith	Ann Smith

attribute which indicates whether or not the attribute has that value. Rules are formed that state

If a set of attributes is true, then another set of attributes is also true.

As all combinations of Boolean attributes have to be considered, the time complexity of the algorithm is exponential in the number of attributes. However, in practice the algorithm takes only 20 seconds for 100,000 tuples [1].

Other fast learning algorithms exploit given hierarchies of attribute values and generalize by climbing the hierarchy (Michalski, 1983), merging tuples that become identical and dropping attributes with too many distinct values to be generalized. The result is a set of rules that characterize all tuples that have a certain value of attribute A in terms of generalized values of other attributes (Cai et al., 1991). Similarly, the KID3 algorithm discovers dependencies between values of two attributes using hierarchies from background knowledge (Piatetsky-Shapiro, 1991). The result is a set of rules of the form

$$A = a' \rightarrow cond(B)$$

where a' is a generalized attribute value (i.e., it covers a set of attribute values) of attribute A and *cond* is some condition of attribute B.

It is easy to see that more complex dependencies between several attributes cannot be expressed (and, hence, cannot be learned) by these fast algorithms. In particular, relations between different tables cannot be learned, since the universal relation for real-world databases is far too big to be computed and stored within sensible time and space limits. Saso Dzeroski has given a nice example to illustrate this (Džeroski, 1996).

For example, from the two tables in Table 1, the fast and simple learning algorithm could discover the rules:

$$income(Person) \geq 100,000 \rightarrow customer(Person) = yes$$

$$sex(Person) = f \;\&\; age(Person) \geq 32 \rightarrow customer(Person) = yes$$

Rules like the following cannot be expressed (and therefore not learned) by these algorithms:

(i) $married(Person, Spouse) \;\&\; customer(Person, yes)$
$\rightarrow customer(Spouse, yes)$

(ii) *married*($Person, Spouse$) & *income*($Person, \geq 100,000$)
$\rightarrow$ *customer*($Spouse, yes$)

Hence, the fast learning algorithms trade in expressiveness for the ability to deal with very large data sets.

The other extreme for making KDD feasible is to select a small subset from the data set (sampling) and learn complex rules. This option is chosen by most inductive logic programming (ILP) algorithms which are applied to the KDD problem. The rule learning task has been stated within the ILP paradigm by Nicolas Helft (Helft, 1987) using the notion from logic of minimal models of a theory $\mathcal{M}^+(Th) \subseteq \mathcal{M}(Th)$.

Definition. (Minimal model) An interpretation I is a model of a theory Th, $\mathcal{M}(Th)$, if it is true for every sentence in Th. An interpretation I is a *minimal model* of Th, written $\mathcal{M}^+(Th)$, if I is a model of Th and there does not exist an interpretation I' that is a model of Th and $I' \subset I$.

Rule learning
Given observations $\mathcal{E}$ in a representation language $\mathcal{L}_\mathcal{E}$ and background knowledge $\mathcal{B}$ in a representation language $\mathcal{L}_\mathcal{B}$,
find the set of hypotheses $\mathcal{H}$ in $\mathcal{L}_\mathcal{H}$, which is a (restricted) first-order logic, such that

(1) $\mathcal{M}^+(\mathcal{B} \cup \mathcal{E}) \subseteq \mathcal{M}(\mathcal{H})$ (validity of $\mathcal{H}$)

(2) for each $h \in \mathcal{H}$ there exists $e \in \mathcal{E}$ such that $\mathcal{B}, \mathcal{E} - \{e\} \not\models e$ and $\mathcal{B}, \mathcal{E} - \{e\}, h \models e$ (necessity of h)

(3) for each $h \in \mathcal{L}_\mathcal{H}$ satisfying (1) and (2), it is true that $\mathcal{H} \models h$ (completeness of $\mathcal{H}$)

(4) There is no proper subset $\mathcal{G}$ of $\mathcal{H}$ which is valid and complete (minimality of $\mathcal{H}$).

This learning task has been taken up by several ILP researchers, among them (Kietz & Wrobel, 1992, Flach, 1992, De Raedt & Bruynooghe, 1992). It is more difficult than the concept learning task.

Concept learning
Given positive and negative examples $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ in a representation language $\mathcal{L}_\mathcal{E}$ and background knowledge $\mathcal{B}$ in a representation language $\mathcal{L}_\mathcal{B}$,
find a hypothesis H in a representation language $\mathcal{L}_\mathcal{H}$, which is a (restricted) first-order logic, such that

(1) $\mathcal{B}, \mathcal{H}, \mathcal{E}^+ \not\models \Box$ (consistency)

(2) $\mathcal{B}, \mathcal{H} \models \mathcal{E}^+$ (completeness of H)

(3) $\forall e^- \in \mathcal{E}^- : \mathcal{B}, \mathcal{H} \not\models e^-$ (accuracy of H)

If the negative examples for concept learning are created using the closed world assumption, all hypotheses that satisfy the three conditions of concept learning also satisfy the first condition of rule learning. Stephen Muggleton and Luc De Raedt have shown this for hypothesis languages that consist of definite clauses (Muggleton & De Raedt, 1994). For $\mathcal{L}_{\mathcal{H}}$ being definite clauses so that there exists a unique minimal model $\mathcal{M}^+(Th)$, condition (2) and (3) of concept learning can be reformulated as:

(2) all $e \in \mathcal{E}^+$ are true in $\mathcal{M}(\mathcal{B} \cup \mathcal{H})$ (completeness of $\mathcal{H}$)

(3) all $e \in \mathcal{E}^-$ are false in $\mathcal{M}(\mathcal{B} \cup \mathcal{H})$ (accuracy of $\mathcal{H}$)

Uwe Kietz has generalized this result and has shown that it is possible that all results of concept learning could also be found by rule learning, but not vice versa. Hence, rule learning is more complex than concept learning (Kietz, 1996)[2]. The difficult rule learning task of ILP also goes beyond the task of knowledge discovery as stated by Heikki Mannila (Mannila, 1995):

Knowledge discovery

Given a database $\mathcal{E}$ and a representation language for hypotheses $\mathcal{L}_{\mathcal{H}}$
find interesting characterizations of the data $\mathcal{H} \subseteq \mathcal{L}_{\mathcal{H}}$ where the interestingness is evaluated by a predicate p such that

(1) $\mathcal{M}^+(\mathcal{E}) \subseteq \mathcal{M}(\mathcal{H})$

(2) $p(\mathcal{E}, \mathcal{H})$ is true.

Background knowledge is not taken into account in Mannila's definition and $\mathcal{L}_{\mathcal{E}}$ is set to databases. However, it is possible to enter background knowledge into the database in the form of additional tables, if it is restricted to ground facts – a restriction which is common in ILP approaches. The real difference between the KDD task and the ILP rule learning task is that Mannila does not demand necessity, completeness and minimality of $\mathcal{H}$. Instead, he demands the interestingness of $\mathcal{H}$. For instance, the predicate in the head of each clause $h \in \mathcal{H}$ could be required to correspond to a database attribute that is considered interesting by the user. ILP algorithms which have a declarative bias can express this condition. The rule learning task is solved by some systems (e.g., RDT (Kietz & Wrobel, 1992), CLAUDIEN (De Raedt & Bruynooghe, 1993), and INDEX (Flach, 1993)). For the application to databases the selected tuples are re-represented as (Prolog) ground facts. In general, ILP algorithms trade in the ability to handle large data sets for increased expressiveness of the learning result.

Given the trade-off between feasibility and expressiveness, we propose a multi-strategy approach. The idea is to combine different computational strategies for the same inferential strategy (here: induction) [3]. The overall learning task is decomposed into a sequence of learning tasks. Simpler subtasks of learning can then be performed by simpler (and faster) learning methods. The simpler algorithms induce hierarchies of attributes and attribute values that structure the hypothesis

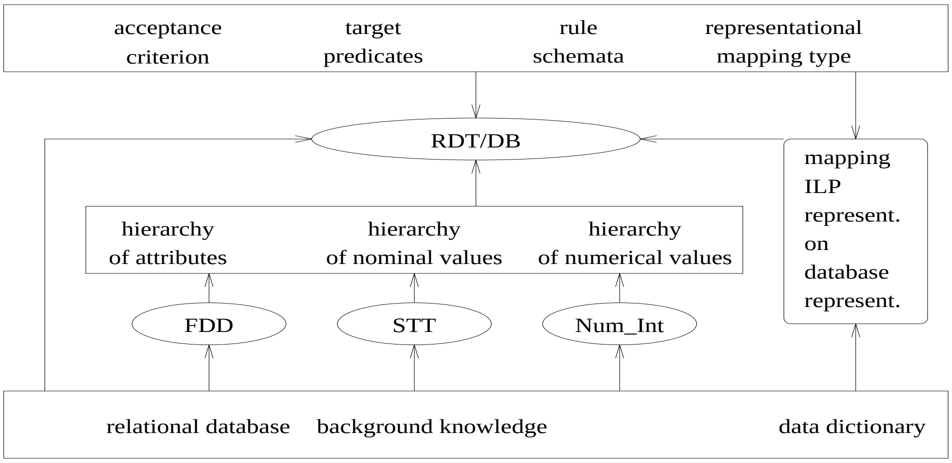

Figure 1. A multistrategy architecture for KDD

space for the ILP learner. The ILP learning algorithm uses this structure for its level-wise refinement strategy. The architecture of our MSL-system for KDD is shown in figure 1.

The paper is organized as follows. First, RDT/DB is described. It is shown how different hypothesis languages $\mathcal{L}_{\mathcal{H}}$ can be created for the same database. $\mathcal{L}_{\mathcal{H}}$ determines the hypothesis space. We analyze the sizes of hypothesis spaces. The analysis shows, where to restrict hypothesis spaces further in order to make learning from very large data sets feasible. Second, the algorithms that preprocess data for RDT/DB are characterized, namely FDD, NUM_INT, STT. Third, we present the results of some experiments and discuss our approach with respect to related work.

2. Applying ILP to Databases

ILP rule learning algorithms are of particular interest in the framework of KDD because they allow the detection of complex rules such as (i), (ii) presented above. Until now, however, they have not been applied to commonly-used relational database systems. If a relational algebra is used, as in the system ML SMART, it is under the control of the learning system, which creates new tables for intermediate results and stores them in the database (Bergadano et al., 1991). Data management and the representation language are tailored for learning. This is particularly appropriate for knowledge acquisition tasks. Since the demand of KDD is to analyze the databases that are in use, we have now enhanced RDT to become RDT/DB, the first ILP rule learner that directly interacts with a commercial database management system.

2.1. RDT/DB

In order to restrict the hypothesis space, RDT/DB uses a declarative specification of the hypothesis language, just as RDT does (see for details (Kietz & Wrobel, 1992)). The specification is given by the user in terms of rule schemata or rule models. A rule schema is a rule with predicate variables (instead of predicate symbols). In addition, arguments of the literals can be designated for learning constant values. A simple example of a rule schema is:

$$mp\_two\_c(C, P1, P2, P3) : P1(X1, C) \ \& \ P2(Y, X1) \ \& \ P3(Y, X2) \rightarrow P1(X2, C)$$

Here, the second argument of the conclusion and the second argument of the first premise literal is a particular constant value that is to be learned. It is indicated in the metapredicate that one constant, C, is to be learned.

For hypothesis generation, RDT/DB instantiates the predicate variables and the arguments that are marked for constant learning. A fully instantiated rule schema is a rule. An instantiation is, for instance,

(iii) $regions(X1, europe) \ \& \ licensed(Y, X1) \ \& \ produced(Y, X2)$
$\rightarrow regions(X2, europe)$

In the example, it was found that the cars which are licensed within Europe have also been produced within Europe.

The rule schemata are ordered by generality: For every instantiation of a more general rule schema there exist more special rules as instantiations of a more special rule schema, if the more special rule schema can be instantiated at all. Hence, the ordering of rule schemata reflects the generality ordering of sets of rules. This structure of the hypothesis space is used while doing breadth-first search for learning. Breadth-first search allows the safe pruning of branches of sets of hypotheses that already have too little support to be accepted. If a rule is learned its specializations w.r.t. the generality ordering of rule schemata will not be tried, since this would result in redundant rules. Hence, RDT/DB delivers most general rules. The user writes the rule schemata in order to restrict the hypothesis space. The user also supplies a list of the predicates that can instantiate predicate variables. This list need not consist of all predicates, but can be a selection.

Another kind of user-given control knowledge is the acceptance criterion. It is used to test hypotheses. The user can compose an acceptance criterion out of 4 terms: $pos(H)$, the number of supporting tuples; $neg(H)$, the number of contradicting tuples; $concl(H)$, the number of all tuples for which the conclusion predicate of the hypothesis holds; and $negconcl(H)$, the number of all tuples for which the conclusion predicate does not hold, although it is applicable. The user can use the criterion to enforce different degrees of reliability of the learning result, or to put it the other way around, to allow different degrees of noise.

A typical acceptance criterion which is similar to that of APRIORI is:

$$\frac{pos(H)}{concl(H)} - \frac{neg(H)}{concl(H)} \geq 0.8$$

The acceptance criterion can also be written in a Bayesian manner. If there are two classes for the conclusion predicate (e.g., faulty and not faulty cars), the following criterion is similar to the requirement that the a posteriori probability must equal or exceed the a priori probability:

$$\frac{pos(H)}{pos(H)+neg(H)} \geq \frac{concl(H)}{concl(H)+negconcl(H)}$$

For RDT/DB we have developed an interaction model between the learning tool and the ORACLE database system[4]. The data dictionary of the database system contains information about relations and attributes of the database. This information is used in order to map database relations and attributes automatically to predicates of RDT's hypothesis language. Note that only predicate names and their arity are stored in RDT/DB as predicate declarations, not a transformed version of the database entries. Hypothesis generation is then performed by the learning tool, instantiating rule schemata in a top-down, breadth-first manner. For hypothesis testing, SQL queries are generated by the learning tool and are sent to the database system[5]. For instance, the number of supporting tuples, $pos(H)$, for the rule (iii) above is determined by the following statement:

```
SELECT COUNT (*)
       FROM  vehicles veh1, vehicles veh2,
             regions reg1, regions reg2
       WHERE reg1.place = veh1.produced_at
             and veh1.ID = veh2.ID
             and veh2.licensed = reg2.place
             and reg1.region = 'europe'
             and reg2.region = 'europe';
```

The number of contradicting tuples, $neg(H)$, is determined by negating the condition which corresponds to the rule's conclusion:

```
SELECT COUNT (*)
       FROM  vehicles veh1, vehicles veh2,
             regions reg1, regions reg2
       WHERE reg1.place = veh1.produced_at
             and veh1.ID = veh2.ID
             and veh2.licensed = reg2.place
             and reg2.region = 'europe'
             and not reg1.region = 'europe';
```

Rule (iii) and the SQL queries refer to a database with two relations as given in Table 2.

The counts for $pos(H)$, $neg(H)$, $concl(H)$, and $negconcl(H)$ are used for calculating the acceptance criterion for fully instantiated rule schemata.

Table 2. Relational database with two tables: vehicles and regions

vehicles:			regions:	
ID	produced	licensed	place	region
fin_123	stuttgart	ulm	ulm	europe
fin_456	kyoto	stuttgart	stuttgart	europe
...	...	...	kyoto	asia
			...	...

As this example implies, RDT/DB can handle negative examples, but RDT/DB is also capable of learning from positive examples alone. Negative examples are either explicitly stored in the database[6] or are there implicitly and are computed by the SQL 'view' statement. In our vehicle application, "negative" facts are, for instance, those cars which do not have a fault. They are constructed by a view which computes the difference between the table with information about all cars and the table with information about cars which have a fault. Since the result of every view is a "virtual table", it can be handled in the same way as every table. It is up to the user to state that $notfaulty(ID) \cong not(faulty(ID))$. That is, the user may declare attributes to have opposite meanings. Obviously, negative literals are not restricted to appear only in the conclusions, but can also appear in the premises of rules.

2.2. Analysis of the Hypothesis Space

The size of the hypothesis space of RDT/DB does not depend on the number of tuples, but on the number of rule schemata, r, the number of predicates that are available for instantiations, p, and the maximal number of literals of a rule schema, k. For each literal, all predicates have to be tried. Without constant learning, the number of hypotheses is $r \cdot p^k$ in the worst case. As k is usually a small number in order to obtain understandable results, this polynomial is acceptable. Constants to be learned are very similar to predicates. For each argument marked for constant learning, all possible values of the argument (the respective database attribute) must be tried. We write c for the number of constants that are marked for learning in a rule schema. Let i be the maximal number of possible values of an argument marked for constant learning; then, the hypothesis space HS is limited by:

$$HS \leq r \cdot (p \cdot i^c)^k$$

Due to the language bias the Vapnik–Chervonenkis Dimension — for first–order logic infinite — becomes finite:

$$\mathcal{VC}dim(HS) \leq log_2 \left(r \cdot (p \cdot i^c)^k \right)$$

This indicates the expressiveness of the hypothesis language. The run–time behavior of rule learning using the declarative language bias can be estimated as follows: The size of the hypothesis space determines the cost of hypothesis generation. For each hypothesis, two SQL statements have to be executed by the database system. These determine the cost of hypothesis testing.

The size of the hypothesis space is described in terms of the representation RDT/DB uses for hypothesis generation. The particular figures for given databases depend on the mapping from RDT/DB's representation to relations and attributes of the database. Therefore, it is important to examine carefully the mapping chosen.

An immediate mapping is to let each database relation become a predicate, the attributes of the relation becoming the predicate's arguments.

Mapping 1: For each relation R with attributes $A_1, \ldots, A_n$, a predicate $rn(A_1, \ldots, A_n)$ is formed, rn being the string of R's name.

For the small database in Table 2, we would have two predicates,
$vehicles(ID, Produced, Licensed)$ and
$regions(Place, Region)$.
If we do not know which of the attributes is relevant, we have to write several rule schemata. Here is an example:
$mp11(P, Q) : P(Y, X1, X2) \rightarrow Q(X1, Z)$
$mp12(P, Q) : P(Y, X1, X2) \rightarrow Q(X2, Z)$
$mp21(P1, P2, Q) : P1(Y, X1, X2)$ & $P2(X2, Z) \rightarrow Q(X1, Z)$
$mp22(P1, P2, Q) : P1(Y, X1, X2)$ & $P2(X1, Z) \rightarrow Q(X2, Z)$
Hypotheses look like these:

(iv) $vehicles(ID, Produced, Place)$
$\rightarrow regions(Produced, Region)$

(v) $vehicles(ID, Produced, Place)$ & $regions(Place, Region)$
$\rightarrow regions(Produced, Region)$

Note, that variables in the conclusion of a rule are universally quantified. If they are not constrained by its premise (as *Region* in (iv)), the rule is very general. It states that one table determines the other one. It is more likely that rules will be found, where the attribute *Region* is a constant to be learned. Hence, when using mapping 1, we most often mark at least one attribute for constant learning. In our example, c could be at most 5, but normally one would exclude the keys, which gives us at most 3 constants to be learned in the example database. All constants in all combinations must be tried. Let i, the number of places, be 2000. Then in our example from Table 2, the hypothesis space has at most $4 \cdot (2 \cdot 2000^3)^2$ hypotheses.

If we map each attribute of each relation to a predicate, we enlarge the number of predicates, reduce the number of rule schemata, and can easily avoid unconstrained universally quantified variables.

Mapping 2: For each relation R with attributes $A_1, \ldots, A_n$, where the attributes $A_j, \ldots, A_l$ are the primary key, for each $x \in [1, \ldots, n] \backslash [j, \ldots, l]$ a predicate $rn\_AX(A_j, \ldots, A_l, A_x)$ is formed, where AX is the string of the attribute name.

If the primary key of the relation is a single attribute, we get two–place predicates. In our example (Table 2), the predicates of the second mapping are

$vehicles\_produced(ID, Produced)$,

$vehicles\_licensed(ID, Licensed)$, and

$regions\_region(Place, Region)$,

where ID and $Place$ are the keys of the tables *vehicles* and *regions*, respectively. Only 2 rule schemata have to be written in order to learn rules which have a similar – but more specific – meaning as rules (iv) and (v):

$mp1(P, Q) : P(Y, X1) \rightarrow Q(X1, Z)$

$mp2(P1, P2, P3, Q) : P1(X1, Z)\ \&\ P2(Y, X1)\ \&\ P3(Y, X2) \rightarrow Q(X2, Z)$

The number of predicates is bound by the number of relations times the maximal number of attributes of a relation (without key attributes). Since the number of constants to be learned cannot exceed the arity of predicates, and since we never mark a key attribute for constant learning, c will be at most k. In our example, we have $2 \cdot (3 \cdot 2000^3)^4$ hypotheses when applying the second mapping.

A third mapping reduces the expressiveness to propositional logic. Of course, this means the size of the hypothesis space is reduced.

Mapping 3: For each attribute A_i which is not a primary key and has the values $a_1, \ldots, a_n$ a set of predicates $rn\_AI\_ai(A_j, \ldots, A_l)$ are formed, $A_j, \ldots, A_l$ being the primary key.

In our example from Table 2 the third mapping delivers these predicates:

$vehicles\_produced\_stuttgart(ID)$

$vehicles\_produced\_ulm(ID)$

...

$vehicles\_licensed\_stuttgart(ID)$

$vehicles\_licensed\_ulm(ID)$

...

$regions\_region\_europe(Place)$

$regions\_region\_asia(Place)$

...

Using this representation, rules like (iii) to (v) cannot be learned. Let the number of different places be again 2000 and the number of regions be 10; then p is 4010. There are no constants to be learned. Hence, the size of the hypothesis space is $r \cdot (4010)^k$. The third mapping reduces the learning task to finding k–place combinations of constant values.

Using the results of Num_Int (cf. section 3.2), which selects intervals of attribute values, a fourth mapping can be applied, which has turned out to be quite powerful. In fact, mapping 3 can be seen as a special case of this mapping where all intervals consist of only one value.

Mapping 4: For each attribute A_i which is not a primary key and for which intervals of values have been computed, $< a_{1p}, a_{1q} >, \ldots, < a_{np}, a_{nq} >$, a set of predicates $rn\_AI < a\_ip, a\_iq > (A_j, \ldots, A_l)$ is formed, $A_j, \ldots, A_l$ being the primary key.

Consider the table *prices* in addition to the tables *vehicles* and *regions* (Table 3).

Table 3. Table *prices*

ID	costs
fin_123	11
fin_456	100
fin_789	150
...	...

A predicate $prices\_costs\_<10,100>(ID)$ would be mapped on the table *prices*, having the attributes *ID* and *costs*. This predicate is true when the values for the attribute *costs* are in the range of 10 to 100. The fourth mapping, which works only for numerical attributes, can be combined with any other mapping for the non-numerical attributes. For instance, the first mapping for the three tables *vehicles*, *regions*, and *prices* creates a hypothesis space of the size $r \cdot (3 \cdot 5000^4)^k$, if we assume 5000 different values for *costs* in the table *prices*. If we have 5 intervals of costs, combining the first and the fourth mapping reduces the size to $r \cdot (7 \cdot 2000^3)^k$, again, 2000 being the number of different places. The fourth mapping allows for a disjunction of values. This does not increase the difficulty of hypothesis testing on the database. The reason this is the case is quite simple. In principal, every predicate in a rule will be mapped onto one table, and all tables will be joined using an equi–join. Having this kind of a predicate, we use instead of an equi–join a θ–join with θ being the expression $costs \geq 10$ and $costs \leq 100$. Instead of using one comparison, e.g. $a = b$, the database system uses two comparisons for predicates of the fourth mapping type. This is only a small increase in computational time, because the most time consuming part is the join projection itself.

The user is not obliged to choose among these four mappings. If he wants to, he can augment the second through fourth mappings with additional attributes in the predicate, moving them closer to the first mapping, e.g. $rn\_AX(A_j, \ldots, A_l, A_x, A_y)$ or $rn\_AI\_ai(A_j, \ldots, A_l, A_y)$.

By choosing a mapping and a set of rule schemata, the hypothesis space is determined. Rule schemata can be written for all mappings. They restrict the hypothesis space by fixing k and c, and by enforcing equality of head variables with a selected variable in the premise. The depth of a variable can be directly seen in the rule schema. These are important factors of learnability of clauses which are now under control of the user. Whether or not the variable bindings are determinate, another important factor, depends on the data[7].

The systems ML SMART and LINUS (Lavrač & Džeroski, 1994), perform a transformation of first-order logic representation into datalog relations. The mapping there is given by the system and cannot be modified by the user. In contrast, we start from database relations and offer the user the opportunity to declare a function-free first-order logic signature. The mapping from this signature back to the database is then automatically performed by RDT/DB.

The analysis of the hypothesis space reveals good heuristics for writing rule schemata: the most general ones should have only one premise literal and the most specific ones not more than 3, as this keeps k small; there should be as few schemata as possible in order to keep r small; at most one constant should be marked for learning in order to keep c small – i, the number of values in an attribute, can only be reduced by the fourth mapping. The selection of relevant attributes can be done by RDT/DB in the second mapping. If there is a more efficient solution to this problem, then RDT/DB should use mapping 1 where the arity of predicates is reduced. The rule schemata allow the restriction of the hypothesis space to those parts considered interesting. The analysis of its size gives a worst-case estimation. This helps in estimating how long a learning pass may last and in eventually changing the declarative bias or the mapping.

3. Further Control of Complexity

Even given the declarative syntactic bias in terms of rule schemata, the hypothesis space in a real-world application can still be very large. This leads to a need to restrict the number of attributes and attribute values used for learning. Some attributes are of particular interest for the user of the KDD system. For instance, in an analysis of warranty cases, the attribute that expresses whether or not an item is a warranty case is the most relevant one. The user can specify this attribute as the target for learning. However, the user does not know the attributes that determine a warranty case. It is the task of learning to identify them! This is the point where FDD comes into play. FDD learns a partial generality ordering on attributes. A sequence of RDT/DB learning passes is started, each pass using only the most general unexplored attributes of the attribute hierarchy for characterizing the user-given target attributes. The sequence is stopped as soon as hypotheses are found that obey the acceptance criterion. That means that after successfully learning rules in language $\mathcal{L}_{\mathcal{H}_i}$, no new learning pass with $\mathcal{L}_{\mathcal{H}_{i+1}}$ is started. Note that the representational bias in terms of the sequence of $\mathcal{L}_{\mathcal{H}_i}$ is a *semantic* declarative bias (as opposed to the syntactic rule schemata or the language sequences of CLINT (De Raedt, 1992)) and is domain-dependent. Using the output of FDD, RDT/DB adapts its behavior to new data sets.

Reducing the number of values of an attribute can be done by climbing a hierarchy of more and more abstract attribute values. If this background knowledge is not available, it has to be learned. For numerical values, NUM_INT finds a hierarchy of intervals. Since this is a learning result in its own right, it is presented to the user, who selects relevant intervals. These are transformed by RDT/DB into predicates that are used instead of the ones that would have been formed on the basis of the original database attribute values. Hence, p slightly increases, but i decreases a lot.

For nominal values, any fast learning algorithm that is capable of finding clusters within the values of one attribute (i.e., that finds sets of attribute values) could be plugged into our multistrategy framework. The clusters are named and these names become more abstract attribute values replacing the original values. In our current application (see below), we have chosen a different approach. Here, we have background knowledge about some different aspects of the attribute values of a given attribute A in the database. The background knowledge is used for learning a graph, where the nodes contain attribute values of the given attributes and the links establish a subset ordering. Since the sets of attribute values are meaningful for the user, he can select nodes of the graph as pertinent. Each selected node becomes a binary attribute of the database. The new attributes replace the original database attribute A in $\mathcal{L}_{\mathcal{H}_i}$. Again, p increases by the number of selected nodes, but i can well decrease by almost all values of the attribute A.

3.1. Detecting Functional Dependencies — FDD

In the following we assume some familiarity with definitions of relational database theory (see, e.g., (Kanellakis, 1990)). Capital letters like A, B, C denote attributes and letters like X, Y, Z sets of attributes. A functional dependency (FD) $X \rightarrow Y$ is valid if every pair of tuples which agrees in their X values also agrees in their Y values. According to Armstrong's Axioms (Ullman, 1988) and without loss of generality we only regard FDs with one attribute on the right hand side. The discovery of FDs may be visualized as a search in semilattices. The nodes are labeled with data dependencies and the edges correspond to the *more general than* relationship as in (Savnik & Flach, 1993), which implies the partial ordering.

Definition. (More general FD) Let X and Y be sets of attributes such that $X \subseteq Y$, then the FD $X \rightarrow A$ is more general than the dependency $Y \rightarrow A$, and $Y \rightarrow A$ is more specific than $X \rightarrow A$.

In contrast to the notion of a minimal cover in database theory, the algorithm FDD computes a *most general cover*. For example, the set $\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$ is most general in our sense, but not minimal according to database theory, since the transitivity rule is applicable. More formally:

Definition. (Most general cover) The set of functional dependencies F of relation R ($R \models F$) is a most general cover, if for every dependency $X \rightarrow A \in F$, there does not exist any Y with $Y \subset X$ and $Y \rightarrow A \in F$.

The hypothesis generation is a top–down, breadth–first search through the semilattice imposed by the *more general than* relationship as in (Mannila & Räihä, 1994). Although the time complexity of FDD has to be exponential in the worst case (Beeri et al., 1984), in practice, FDD successfully learned from several databases of the size of 65 000 tuples in up to 6 minutes. This fast implementation differs from the one of (Mannila & Räihä, 1994) in four points that are minor but nevertheless contribute to its efficiency. First, null values of the database are taken into account. Second, the grouping operation of the database system replaces the learning algorithm's sorting of tuples. Third, a pretest of whether all attributes minus one determine this attribute is used for pruning. Fourth, while Mannila and

Table 4. A SQL query for the computation of functional dependencies, ($B \notin \{A_1 \ldots A_n\}$)

```
SELECT MAX (COUNT (DISTINCT B))
FROM R
GROUP BY A1, ..., An                =:a1
```

$a_1 = 1 \Rightarrow A_1 \ldots A_n \rightarrow B$

Räihä merely exploit the transitivity of functional dependencies, here functional independencies are also used (Bell, 1995).

FDD uses the interaction model that was described above, i.e. FDD generates SQL queries (i.e. hypotheses) and the database system computes the answer (i.e. tests the hypothesis). Table 4 shows an example of this kind of statement and the condition which must hold. The clue to detecting functional dependencies is the GROUP BY instruction. The computational costs of this operation are dependent on the database system, but it can be done in time $\mathcal{O}(m * log\, m)$, where m is the number of tuples.

We define a hierarchy on the involved attributes of unary FDs in the following way:

Definition. (More general attribute) Given a set of unary functional dependencies F, the attribute C is more general than A if $A \rightarrow C$ is an element of the transitive closure of F and $C \rightarrow A$ is not an element of the transitive closure of F.

We present a simple example to illustrate this. Let the only valid FDs in R be the following: $\{A \rightarrow B, B \rightarrow C\}$. Then we will get a hierarchy of attributes, where C is more general than B, and B more general than A. Since the three attributes are from the same relation and the FDs hold, there must be at least as many tuples with the same value for C as for B. This follows immediately from the definition of FDs. The same is true for B with respect to A. Furthermore, if there are cycles in the transitive closure of these unary FDs, then all attributes within the cycle are of equal generality. Attributes entering the cycle are more specific than the ones within it. Attributes leaving the cycle are more general than attributes within it.

We exploit this *more general* relationship on attributes for the development of a sequence of hypothesis languages. Each stays within the same syntactical structure as given by the rule schemata, but only has a subset of the attributes. Given, for instance, the FDs $\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$, the first set of predicates in $\mathcal{L}_{\mathcal{H}_1}$ includes C and neither A nor B, $\mathcal{L}_{\mathcal{H}_2}$ includes B and neither A nor C, and $\mathcal{L}_{\mathcal{H}_3}$ includes A and neither B nor C. As a result, we have a level–wise refinement strategy as in (Mannila & Toivonen, 1996), which means, we start the search with hypotheses consisting of most general attributes. If these hypotheses are too general, we continue with only more specific attributes.

3.2. Discretization of Numerical Attributes — NUM_INT

Numerical values offer an ordering that can be exploited. Hence, these values can be grouped with less complexity than nominal values, even if no classification is available. The idea behind NUM_INT is to order the numbers and to search for "gaps" in the stream of values. The biggest gap is supposed to be the best point for splitting up the initial interval [min, max]. The next gaps are taken to continue splitting in a top–down fashion. The result is a tree with the initial interval [min, max] as the root, split intervals as inner nodes and unsplit intervals as leaves. The depth of this tree is determined by a parameter (d) which is set by the user.

The result is obtained by three conceptual steps: first, order the numbers (using the statement "select ... order by ..." via embedded SQL); second, fetch tuple by tuple (via embedded SQL), gathering all information needed for splitting; third, build up the tree of intervals. The complexity of the three steps is as follows: step (1) should be $\mathcal{O}(m \log m)$, m being the number of tuples, because we select only one attribute (this is done by the database system and therefore beyond our control). In step (2) each gap has to be sorted into an array of depth d, which leads to $\mathcal{O}(m \cdot d)$. Finally, in step(3) we have to insert $\mathcal{O}(d)$ values into an ordered array of depth $\mathcal{O}(d)$ resulting in complexity of $\mathcal{O}(d^2)$.

Most of the time is consumed by simply fetching the tuples one by one via SQL. We tested NUM_INT on a database containing about 750.000 tuples: the algorithm ran 35 minutes on a SUN SPARC 20 for a depth of 100. Of this, 2 minutes were used for ordering, 8 minutes for internal processing and about 25 minutes for waiting on the database system to deliver the tuples. This shows that it is essential that we touch each tuple only once and collect all information (min, max, found intervals, number of tuples in each interval) "on the fly".

Of course, we are aware of the fact that this "gap"–approach is a quite simple one and that more sophisticated approaches for learning intervals are known (e.g., (Wettscherek & Dietterich, 1995)). Pazzani, for instance, presented an iterative improvement approach for finding discretizations, i.e. intervals, of numeric attributes (Pazzani, 1995). His algorithm starts with a partition of the input numeric values into a small number of seed intervals with equal size, and then iteratively uses split or merge operations on the intervals based on error or misclassification costs. In most cases, an appropriate number of intervals is unknown in advance, resulting in some loops in the algorithm in which he has to reconsider all values of the attribute. In particular, clustering algorithms, although much more elegant, are too complex to be applicable.

3.3. Forming a hierarchy of nominal attribute values – STT

Background knowledge is most often used in a KDD framework in order to structure sets of attribute values, that is, the background knowledge offers a hierarchy of more and more abstract attribute values. However, background material is often unstructured. When this is so, it needs some structuring before it can be used for learning

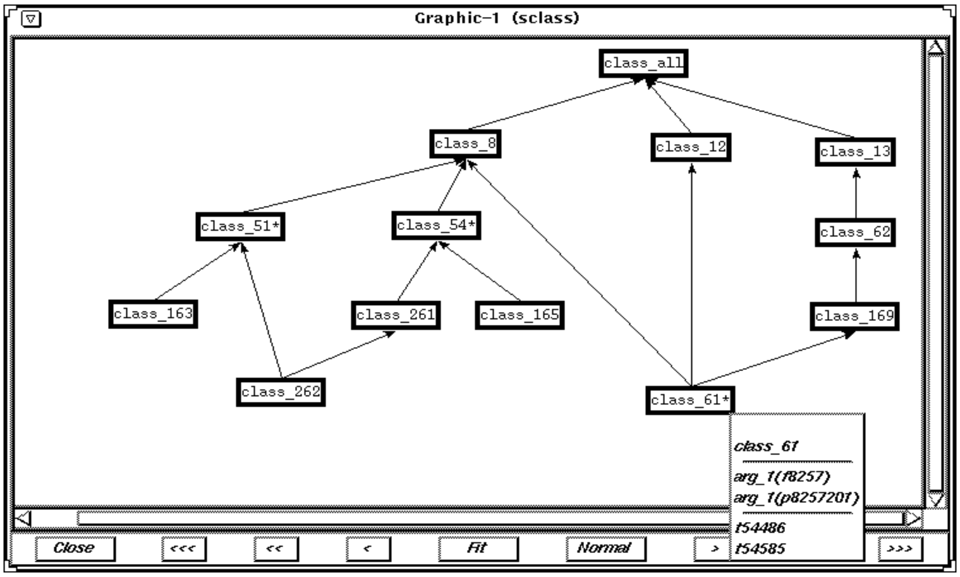

Figure 2. Part of the sort lattice computed by STT.

from the database. For this task, we use STT, a tool for acquiring taxonomies from facts (Kietz, 1988). First, it looks at each predicate and forms value sets consisting of the constant values which occur at each argument position. For instance, the constant values a, b, c may occur at the first position of predicate p_1. Second, it computes equivalence classes between the extensions of arguments. For instance, if all values occurring as the first argument of p_1 also occur as the second argument of p_2 and vice versa, an equivalence class for the set of these values is formed and given a unique name: $class1$ consists of $ext(arg\_1(p_1)), ext(arg\_2(p_2))$, where the extension of the two argument positions is equal, namely $\{a, b, c\}$. Equivalence classes are built for all value sets. If a value set occurs uniquely as the argument of one predicate, the respective class has just one member. For instance, let the extension of the first argument of predicate p_2 occur uniquely: then $class4$ consists of $ext(arg\_1(p_2))$. Finally, the classes are partially ordered. A class c_1 is a subclass of a class c_2 if the extension of elements in c_1 is a subset of the extension of elements in c_2. It may turn out, for instance, that all values that occur as second argument of a predicate p_1 also occur as first argument of predicate p_3, but not the other way around. In this case, the class $class3 : ext(arg\_2(p_1))$ becomes a subclass of $class2 : ext(arg\_1(p_3))$. We omit a detailed and formal presentation of STT here and refer to chapter 4 in (Morik et al. 1993). The point here is that STT can effectively group together attribute values on the basis of background knowledge.

We have represented textual background material about Mercedes car parts as unary ground facts. STT forms classes on the basis of the given facts. The resulting semilattice of classes is interesting in its own right. It can easily be displayed graphically as shown in Figure 2. For each class, its members and their extension can be displayed, as well. In Figure 2, the members of $class\_61$ ($arg\_1(f8257)$, $arg\_1(p8257201)$) are shown together with their extension ($t54486, t54585$). The numbers $t54486, t54585$ denote car parts, the predicate $f8257$ a functional group of parts and the predicate $p8257201$ a spatial group of car parts. Predicates as well as arguments are immediately understandable to the domain experts. This allows the user to select interesting classes. These are introduced as Boolean attributes into the database. This increases p, but decreases i dramatically, since the classes are used for learning instead of using several thousands of attribute values in the original database.

4. Experiments

In the course of an on–going project at Daimler Benz AG on the analysis of their data about vehicles, we have applied our multistrategy learning. The database with all vehicles of a certain type of Mercedes — among them some cases involving warranty — is of size 2.6 gigabytes. It consists of 40 relations with up to 40 attributes each. The main topic of interest for the users is to find rules that characterize warranty cases and structure them into meaningful classes. In a monostrategy approach, RDT/DB could well find rules, among them ones that are about 100% correct. However, these rules were not interesting, since they expressed what is known to all mechanics. For instance, it was learned that an engine variant determines the engine type. This is exactly what is meant by *variant* and *type*.

(vi) $engine\_variant(VIN, 123) \rightarrow engine\_type(VIN, 456)$

More interesting rules could only be found by preprocessing the data and thereby focusing RDT/DB on interesting parts of the hypothesis space. FDD selected sets of attributes for a learning pass. NUM_INT found intervals in the cost attribute of warranty cases. This made rule learning feasible for a very large database. The introduction of the new attributes on the basis of STT's output led to the learning of more interesting rules. The background material is the mechanic's workbook of vehicle parts, classified by functional groups of parts, spatial groups (a part is close to another part, though possibly belonging to another functional group), and possible faults or damages of a part. The vehicle parts are numbered. *t54486*, for instance, is a certain electric switch within the automatic locking device. The functional groups of parts are also numerically encoded. *f8257*, for instance, refers to the locking device of the vehicle. The fact *f8257(t54486)* says that the switch *t54486* belongs to the locking device. The spatial grouping is given by pictures that show closely related parts. The pictures are, again, numerically encoded. *p8257201*, for instance, refers to a picture with parts of the electronic locking device that are closely related to the injection complex. *p8257201(t54486)* tells that the switch belongs to the spatial group of the injection. Possible damages or faults depend, of course, on the material of the part rather than on its functional group. All different types of damages are denoted by different predicates (e.g., *s04* indicates that the part might leak). The combination of these three aspects has led to STT finding some surprising classes. Looking at Figure 2, class_61 comprises two switches, *t54486* and *t54585*. They are the intersection of three meaningful classes:

class_169 : here, several parts of the injection device are clustered. These are parts such as tubes or gasoline tank. Higher up in the hierarchy, parts of the functional group of injection and then (class_13) parts of gasoline supply in general are clustered.

class_12 : here, parts of the dashboard are clustered, among them the display of the locking device (protection from theft).

class_8 : here, operating parts of the engine are clustered that serve the injection function.

The illustration shows that mechanics can easily interpret the clusters of parts, and that therefore the hierarchy learned by STT is meaningful. The intersection classes are very selective, whereas classes such as, e.g., class_13 cover all parts of a functional group (here: gasoline supply). Domain experts are particularly interested in finding groups that are one level below the functional groups that are documented in the mechanic's workbook.

We have conducted several experiments in order to answer two questions:

- Are the results of a multistrategy approach superior to the results of a rule discovery tool?
- Can an ILP rule learning algorithm handle the mass of data that is given by real-world databases; or is statistical sampling a necessary prerequisite?

We made four experiments on a Sparc 20 computer[8]. The real-world data are characterized in Table 5. All data are about a class of Mercedes cars which is no longer produced.

The first sample of data is a transformed version of database tuples. This was our first experiment with RDT and motivated us to develop RDT/DB. Although the data do not show the effect of directly accessing the database management system, they are listed here because they show the effect of multistrategy as opposed to monostrategy learning.

The size of meaningful and relevant samples of the original data is still rather large. For instance, such samples might include data about warranties of cars with gasoline engine and automatic gearshift, of cars with gasoline engine and manual gearshift, and of cars with diesel engine and automatic gearshift. They still consist of 23 database tables with 1 to 3 attributes in each table selected for learning and 111,995 tuples. If we would translate the database to ground

Table 5. Summary of data sets
Meaning of columns:
t — no. of tables
j — max. no. of tables in a join projection
attributes — no. of attributes in each table
i — max. no. of attribute values (excluding keys)
tuples — max. no. of tuples in a table
facts — no. of equiv. facts

Data sets	t	j	attributes	i	tuples	facts
excerpt	–	–	–	–	–	717
all cars1	1	0	1	700,000	750,000	750,000
all cars within 3 months	7	0	2 – 14	7,869	43,194	771,162
gas, manual	23	3	1 – 3	166	111,995	39,000
gas, autom.	23	3	1 – 3	166	111,995	2,080,000
diesel, autom.	23	3	1 – 3	166	111,995	130,000
car parts	–	–	604 preds.	–	–	20,794
gas, manual +	16	8	1 – 9	166	700,000	57,351
all cars	3	3	3 – 9	700,000	750,000	1,500,820

facts, we get the following figures. One observation (or example) is made of 26 ground facts: The target predicate stating whether it is a case of warranty, 11 facts on the configuration of the car, and up to 14 special equipment statements. For gasoline engine and manual gearshift, 1500 observations are available. For gasoline engine and automatic gearshift, 80,000 observations are available. For diesel engine and automatic gearshift, 5000 observations are available. This explains how we calculated the number of facts that are equivalent to the database selection. We have included the maximal number of tuples in the description of the data, although it is not the size of the attribute on which we wanted to learn, since it gives an impression of how large the set of tuples is in which the database system must search for a characterization of warranty cases.

Another feature of the data with respect to the learning tasks is the number of join projections that are necessary for putting together an observation (or case). This seems to have considerable influence on the time spent for learning. It does not determine the complexity of hypothesis generation (the size of the hypothesis space), but it does contribute to the complexity of hypothesis testing using a given real-world database system.

The results of the first three experiments are shown in Table 6. The first experiment compares the results of RDT alone (pass 0) with the results of RDT in concert with NUM_INT and FDD (pass 3). The learning passes of NUM_INT and FDD are also characterized (passes 1 and 2). The column *conclusion predicates* needs some explanation. The hypothesis language $\mathcal{L}_{\mathcal{H}}$ can be restricted such that only 1 predicate can instantiate the predicate variable in the conclusion. To be fair, we indicate how many predicates can instantiate the conclusion predicate and for how many of these instantiations rules were found. For instance, 5 of 9 means that 9 predicates were instantiated and for 5 of them rules were learned. The first experiment shows that NUM_INT increased the validity of the learned rules. FDD

Table 6. Summary of learning results

Learning passes	algorithms	data sets	concl. predicates	a priori prob.	no. of learned rules	average of a posteriori prob.	time spent
Pass 0	RDT	excerpt	5 of 9	–	67	33-63%	28min.
Pass 1	Num_Int	all cars1	–	–	–		35min.
Pass 2	FDD	all cars within 3 months	108	–	246 FDs	–	6min. 5sec.
Pass 3	RDT, Num_Int, FDD	excerpt + costs	5 of 9	–	63	79-84%	24min.
Pass 4	RDT/DB	gas, manual	1 of 1	48.15 %	17 (10)	56.2% (61.8%)	3min. 15sec.
		gas, autom.	1 of 1	84.48 %	79 (74)	94.67% (95.35%)	3h. 1min. 46 sec.
		diesel, autom.	1 of 1	99.11%	15 (3)	99.13% (99.24%)	7min. 29sec.
Pass 5	STT	car parts	–	–	498 classes	–	12h.
Pass 6	RDT/DB, STT	gas, manual+	4 of 9	9.82%	92	15.47%	49h. 32min. 44sec.
Pass 7	RDT/DB, Num_Int, STT, FDD	all cars	6 of 9	62.38%	7	68.83%	13h. 30min

contributed by reducing the set of predicates in $\mathcal{L}_{\mathcal{H}}$. Hence, RDT learned more quickly and also learned more accurate rules.

The second experiment is made of a sequence of learning passes. First, RDT/DB learns to characterize warranty and non-warranty cases (pass 4). Background knowledge about special equipment of cars is used in order to make the learning results more interesting. For instance, the rule

(vii) $rel\_niveaureg(VID) \rightarrow faulty(VID)$

states that cars with the special equipment of "niveau regulation" are more likely to be warranty cases than all other cars taking into account the distribution of cars with this equipment. Other rules hint at the influence of an engine variant and the number of cylinders.

(viii) $motor\_e\_type(VID, Type)$ & $mobr\_cyl(Type, 6.0) \rightarrow faulty(VID)$

The size of the hypothesis space is the same for all three data sets. The mapping used is also the same. Mapping 1 with selected attributes is used for most of the database tables. This means the arity of the corresponding predicates has been reduced to relevant ones. Mapping 3 is used for the regions and classes of car parts. The conclusion predicate is instantiated and the premise consists of no more than two literals. For k, the conclusion literal need not be counted. The size of the hypothesis space, therefore, is $4 \cdot (26 \cdot 166^1)^2$. Although it is the same for the three data sets, the time spent for learning varies considerably. The reason for this is the different opportunity for pruning which is indicated by the number of learned rules. We used the Bayes acceptance criterion for two classes, which means that the a posteriori probability should be greater than or equal to the a priori probability. The number of learned rules with greater probability is shown in brackets. Within the first data set it was safe to prune after finding 17 rules. In the second data set, many more rules could be found with high a posteriori probabilities, i.e., there were many factors to be found that characterize warranty cases. Hence, learning time increased as well as the number of learned rules. Pruning again shortened the learning time in the third data set.

Since the findings on gasoline cars with manual gearshift were not exciting, as a second step we ran STT on all car parts and found 498 classes that combine functional and spatial aspects as well as types of damage (pass 5). We selected 9 classes and constructed them as Boolean attributes in an additional database table. The enhanced data for gasoline cars with manual gearshift describe 11 configurations of cars, the class of the car part that was faulty, and up to 4 kinds of special equipment (i.e. we left out that equipment that never occurred in a learned rule). Background knowledge about the regions of production remained in the data set. One observation consists of 21 statements about the car and its parts. Since one car part can be in the extension of several part classes, we get indeterministic variable bindings.

The third step of the experiment was running RDT/DB again, this time using the results of the other learning algorithms (pass 6). It used 13 metapredicates with up to 7 literals. Hence, the size of the hypothesis space was enormous: $13 \cdot (21 \cdot 166^2)^7$. This is a challenging size and we wondered whether learning could be accomplished at all. RDT/DB discovered rules that characterize 4 of the 9 part classes. Two examples of rules found are the following:

(ix) $rel\_warranty(X1, X2, RB, X4, X5, X6, X7, Config, Part)$ &
$rel\_warranty\_gasman(X1, X2, RB, X4, X5, X6, X7, VID)$ &

$rel\_motor(VID, Type, Variant)$ & $engine\_perf(Type, Variant, 236.0)$
$\rightarrow class\_15(Config, Part)$

(x) $rel\_warranty(X1, X2, RB, X4, X5, X6, X7, Config, Part)$ &
$rel\_warranty\_gasman(X1, X2, RB, X4, X5, X6, X7, VID)$ &
$rel\_motor\_type(VID, 206)$ & $regions\_italy(RB)$
$\rightarrow class\_419(Config, Part)$

The first rule states that having difficulties with the tuning of the fuel injection process (class 15) is a reason for a warranty case particularly for cars with an engine performance of 236. The second rule states that warranty claims concerning particular parts of the fuel injection (class 419) are made particularly in Italy and for variant 206 of engines [9]. The rules explain under which circumstances members of a set of car parts are more likely to be faulty than regularly. Because of the appropriate level of granularity that was introduced by STT, the rules are interesting for domain experts.

The third experiment with the multistrategy approach combines RDT/DB, NUM_INT and STT (pass 7). FDD selected interesting, non-redundant attributes. The learning task was to characterize costs of warranty cases in terms of car parts, regions, and car configurations. For every cost interval found by NUM_INT, all other cost intervals were marked as its negative counterpart. The car parts grouped together by STT were constructed as Boolean attributes of an additional table, as before. Again, we have indeterministic variable bindings in the hypotheses, since a car part can be in the extension of several classes. Since no constant was marked for learning the size of the hypothesis space is $1 \cdot (17 \cdot 700,000^0)^3$.

One surprising finding was that parts of class 35 are the prominent class in two of the 9 cost intervals. This is an example of a rule that is unlikely to be found by a concept learner. The rules tell us that parts of *class_35* make two cost intervals more likely than they normally are. It is also interesting that three classes could never be found to be a reason for a sufficiently great number of warranty cases. For all other intervals, the learned rules relate a class of parts to just one interval.

(xi) $rel\_warranty(X1, X2, X3, X4, X5, X6, X7, Config, Part)$ &
$class\_35(Config, Part)$
$\rightarrow cost\_ < 0\_500 > (X1, X2, X3, X4, X5, X6, X7)$
a priori prob.:0.84, a posteriori prob.: 0.89

(xii) $rel\_warranty(X1, X2, X3, X4, X5, X6, X7, Config, Part)$ &
$class\_35(Config, Part)$
$\rightarrow cost\_ < 1015\_4960 > (X1, X2, X3, X4, X5, X6, X7)$
a priori prob.:0.05, a posteriori prob.: 0.07

(xiii) $rel\_warranty(X1, X2, X3, X4, X5, X6, X7, Config, Part)$ &
$class\_419(Config, Part)$
$\rightarrow cost\_ < 0\_500 > (X1, X2, X3, X4, X5, X6, X7)$
a priori prob.:0.84, a posteriori prob.: 0.96

Table 7. Learning passes with FOIL and CLAUDIEN

Learning algorithms	data sets	concl. predicates	no. of learned rules	eval.	time spent
FOIL	art. gas, autom.	1	139	73.6% cov. 39.9% acc.	8 h. 5 min.
FOIL	art. gas, manual	1	10	5.6% cov. 99.7% acc.	42 sec.
FOIL	art. diesel, autom.	1	39	99.47% cov. 25% acc.	4 min. 41 sec.
FOIL	art. gas, manual+	9	6	78,2% cov. 99,9% acc.	3 h. 16 min. 22 sec.
CLAUDIEN	cars within 3 months	1	40	10% a priori 12% a posteriori	14 h. 10 min

The learning time in this experiment is greater than we would expect given the rather small hypothesis space. We see here a disadvantage of directly coupling the learning algorithms with the database system. Although the number of tuples needed to verify or falsify a hypothesis is small, the selection of them within tables of 750,000 tuples takes time. Learning algorithms that select and prepare the data before learning never report on the time for preprocessing, which lasts several days. In contrast, all preprocessing is part of our multistrategy approach and its time is included in the learning time.

The set of experiments shown in Table 6 tends to support positive answers to our two questions about multistrategy and handling mass data by ILP rule learning. First, the preprocessing tools introduced new terms into the hypothesis language (classes of STT and intervals of NUM_INT) that enabled RDT/DB to learn rules that otherwise could not have been learned. The results of passes 3, 6, and 7 justify a moderate positive answer to the first question: the multistrategy approach is superior with respect to the validity of the learned rules. Second, many observations consisting of many features can be handled by an ILP rule learning algorithm. With respect to sampling – the most commonly used alternative to directly accessing the given database – additional experiments would be necessary. Here, we wanted to show that sampling is not the only way of coping with mass data.

In order to see whether other ILP algorithms can cope with mass data, we have run FOIL (Quinlan, 1990) and PROGOL (Muggleton, 1995) on large data sets. We generated artificial data that mimic the real-world data, but are stated in the format required by these algorithms[10]. There were a number of ways the task was made simpler. When converting the tuples from the database to tuples for FOIL or ground facts for PROGOL we used the learning results obtained by RDT/DB in that we did not transform irrelevant attributes into the representation. For instance, the database sample of all cars with gasoline engine and automatic gearshift would consist of 2,080,000 ground facts, but we generated 1,300,000 facts for the comparison. This eases the learning task. Moreover, we ran the algorithms on our best Sparc 20 machine with 128 MB of main memory. Passes of RDT/DB were run on the Sparc

20 machine which stores the database. This computer has 96 MB of main memory. In addition, the learning task of FOIL is the simpler one of concept learning. It is learning the definition of the concept *warranty*. If the definition is accurate enough, learning stops. In contrast, the results of rule learning are independent rules. Even if an appropriate concept definition has been found, learning has to continue in order to look for more most-general, valid, and non-redundant rules.

Because of these simplifications, the comparison between the rule learning of RDT/DB and the concept learning of FOIL can only investigate the matter of learning from mass data (cf. Table 7). The evaluation of the FOIL results are given in terms of coverage of positive examples (i.e. warranty cases) and in terms of accuracy. From the four learning passes, three are performed more quickly by FOIL than by RDT/DB, but one is performed more slowly. This is surprising, since the learning task is less complex. It indicates that FOIL is sensitive to the number of tuples. In particular, it is dependent on all tuples fitting into main memory, which was not the case for the data on gasoline engines with automatic gearshift. PROGOL was not able to handle this amount of data at all. It broke down after 10 minutes when trying to learn from the data set of gasoline engines and automatic gearshift. Even with the smallest data set it could not deliver a learning result within 72 hours.

The comparison with another ILP rule learner may be more appropriate. The system CLAUDIEN (De Raedt & Bruynooghe, 1993) was run by Luc Dehaspe and Hendrik Blockeel on a selection of all cars within 3 months minus the table concerning special equipment. The selected data were converted into ground facts. The a priori probability was 10%. The rules found have an average a posteriori probability of 12%. Hence, CLAUDIEN found valid rules. The run-time of the system on the selection of cars (14 hours) lies within the same range as the run-time of RDT/DB in concert with NUM_INT, STT, and FDD when learning from *all* cars.

5. Discussion

In this paper we have presented a multistrategy architecture for learning rules in a restricted first-order logic from very large data sets. The mapping of first-order hypotheses on databases has been discussed with respect to the size of the resultant hypothesis space. This analysis indicates a way in which several additional algorithms can serve a rule learning algorithm. The attributes of the database are ordered by FDD. The rule learning algorithm is applied to only one level of this partial ordering. A sequence of learning passes realizes a stepwise refinement strategy. The values of a database attribute are reduced by NUM_INT, if they are numerical, and by STT, if unstructured background material is available. The combination of the four learning algorithms allows one to learn first-order rules from very large data sets.

Whether the user should select appropriate levels from the learned hierarchies of the "service algorithms" is an issue for discussion. We have adopted the position of (Brachman & Anand, 1996) that the user should be involved in the KDD process. From the point of view of the algorithm, the selection of one layer as opposed to trying all combinations of all hierarchies makes learning feasible even in very large databases. From the point of view of the user, he is interested in preliminary results and wants to have control of the data mining process. The user is interested in some classes of hypotheses and does not want to specify this interest precisely as yet another declarative bias. Note, however, that the user in our framework does not need to implement the interaction between the learning result of one algorithm and its impact on the other algorithm. This is particularly different from the KEPLER KDD workbench, which offers a variety of learning algorithms together with a set–oriented layer for data management (Wrobel et al., 1996). KEPLER allows the user to call various learning tools and use the results of one as input to another one. It is a loosely coupled system, whereas ours is a tightly coupled one.

For the purpose of directly accessing database management systems, RDT was enhanced so that hypothesis testing is executed via SQL queries. Database management systems offer services for mass data that exceed main memory. Since main memory today is rather huge, this advantage has become less important. Sometimes, the database management makes the system pay a certain amount of overhead. For instance, the 35 minutes run-time of NUM_INT consisted of 25 minutes of waiting on the database system. In pass 7 we also experienced this overhead of the database system. However, the very long duration of FOIL's learning in the one case where the tuples do not fit into main memory (5 hours longer than the run-time of RDT/DB) show how much time can be saved by directly accessing a database. More performance tests are necessary in order to clearly determine under which conditions the database management system speeds up or slows down the learning process.

Accessing a given database management system directly is very different from employing a relational algebra for representing observations as ML SMART does (Bergadano et al., 1991). The storage of data cannot be influenced when accessing a given database. The database which has been designed for other needs than learning can only be queried but not modified. It is possible that the same selection of data or even join projection of tables has to be calculated over and over again. In contrast, ML SMART takes advantage of its own data management. ML SMART creates new tables, stores a selection of data as intermediate result, or stores aggregations of data. In the course of learning, the database is changed. This reduces the number of tuples during learning[11].

RDT/DB bridges the gap between the needs of learning and the given data. It uses declarations of mappings from database to learning representation. It checks whether a hypothesis is unsatisfiable. It automatically transforms a declarative bias into database terms. This service corresponds to a lot of preprocessing work that is usually done before running a learning algorithm. Together with the algorithms NUM_INT, STT, and FDD most of the preprocessing is performed by the learning system itself. A true comparison with other learning systems should consider the time spent for preprocessing *and* learning.

The rule learning algorithm RDT is particularly well suited for KDD tasks because its complexity is not bound with respect to the number of tuples, but with respect to the representation of hypotheses. Its top-down, breadth-first search allows large parts of the hypothesis space to be safely pruned. The declarative syntactic bias is extremely useful for restricting the hypothesis space when learning from very large data sets. However, the syntactic language bias is not enough to apply RDT to real-world databases without reducing it to the expressiveness of an algorithm such as KID3, for instance. If we want to keep the ability to do relational learning but also want to learn from all tuples of a large database, we need more restrictions. These restrictions need to lead to a reduction in the number p of predicates or in the maximal number i of attribute values for an attribute. The restrictions should be domain–dependent. The task of structuring the set of attributes as well as the task of structuring sets of attribute values is performed by more specialized learning algorithms.

Right now, we can state that without using various methods in concert, we achieved valid but not interesting results. Some types of relations could not at all be learned without preprocessing by other methods. For instance, no rules involving costs of warranty cases could be found before NUM_INT delivered the intervals. Without the further concepts found by STT, the analysis of reasons for warranty cases would not have been possible. RDT/DB successfully solved the difficult rule learning task on very large data sets. Together with the other learning algorithms it could learn from 750,000 tuples and three database relations. While we are convinced that this work–share between a number of more specialized algorithms and a more general learning algorithm is a powerful idea, we do not claim that the algorithms for structuring attribute values are in general the best choice. However, our architecture allows us to plug in other (better) algorithms, if available.

Acknowledgments

Work presented in this paper has been partially funded by Daimler–Benz AG, Research Center Ulm, Contract No.: 094 965 129 7/0191. We thank G. Nakhaeizadeh and R. Wirth for indicating practical needs and encouraging us to fulfill these needs using relational learning. Christian Franzel has developed the NUM_INT algorithm. Mark Siebert has acquired background knowledge about vehicle parts and has applied STT to it. We also thank J.–U. Kietz for fruitful discussions and the reviewers for stimulating comments.

Notes

1. In (Agrawal et al., 1996) the authors present a series of experiments with the algorithms and give a lower bound for finding an association rule.
2. Proofs for the various representation languages used in ILP can be found in (Kietz, 1996).
3. According to (Michalski, 1994) the *computational strategy* means the type of knowledge representation and the associated methods for modifying it in the process of learning. The *inferential strategy* means the primary type of inference underlying a learning process.
4. A first implementation was presented in (Lindner & Morik, 1995).

5. The learning system RDT/DB and the DBMS Oracle V7 are loosely coupled via a TCP/IP connection.
6. Here, we assume that the user has done this beforehand; we do not generate negative examples by means of the closed–world assumption.
7. For an overview of learnability results of ILP see (Kietz & Džeroski, 1994).
8. All figures for run-time refer to real time, not CPU time. Since we were not the only users, the time spent for learning is greater than it would be in single-user mode.
9. The engine performance value in this rule and the variant number in the preceding rule have been changed.
10. The data sets have the same names as their corresponding one with the prefix 'art.'.
11. Whether the data manipulation of ML SMART is in fact superior for learning compared with that of a commercial database management system is an open question. In (Bergadano et al., 1991), no experiments with large data sets are reported together with the run-time of the system. The analysis of Michael Pazzani and Dennis Kibler (Pazzani & Kibler, 1992) states that ML SMART runs in doubly exponential time. Therefore it seems unlikely that it could be applied to mass data.

References

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining.* Cambridge, MA: AAAI Press/MIT Press.

Beeri, C., Dowd, M., Fagin, R., & Statman, R. (1984). On the structure of Armstrong relations for functional dependencies. *Journal of the ACM, 31*, 30–46.

Bell, S. (1995). Discovery and maintenance of functional dependencies by independencies. In U. M. Fayyad & R. Uthurusamy (Eds.), *Proceedings of the First International Conference on Knowledge Discovery and Data Mining* (pp. 27–32). Cambridge, MA: AAAI Press/MIT Press.

Bergadano, F., Giordana, A., & Saitta, L. (1991). *Machine learning: An integrated framework and its applications.* New York: Ellis Horwood.

Brachman, R. J., & Anand, T. (1996). The process of knowledge discovery in databases: A human-centered approach. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining.* Cambridge, MA: AAAI Press/MIT Press.

Cai, Y., Cercone, N., & Han, J. (1991). Attribute–oriented induction in relational databases. In G. Piatetsky-Shapiro & W. Frawley (Eds.), *Knowledge discovery in databases.* Cambridge, MA: AAAI Press/MIT Press.

De Raedt, L. (1992). *Interactive theory revision: An inductive logic programming approach.* New York: Academic Press.

De Raedt, L., & Bruynooghe, M. (1992). An overview of the interactive concept–learner and theory revisor CLINT. In S. Muggleton (Ed.), *Inductive logic programming.* London: Academic Press.

De Raedt, L., & Bruynooghe, M. (1993). A theory of clausal discovery. In R. Bajcy (Ed.), *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, (pp. 1058–1063). San Mateo, CA: Morgan Kaufmann.

Džeroski, S. (1996). Inductive logic programming and knowledge discovery in databases. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining.* Cambridge, MA: AAAI Press/MIT Press.

Flach, P. A. (1992). A framework for inductive logic programming. In S. Muggleton (Ed.), *Inductive logic programming.* London: Academic Press.

Flach, P. A. (1993). Predicate invention in inductive data engineering. In P. Brazdil (Ed.), *Procs. of the European Conf. on Machine Learning (ECML-93)*, volume 667 of *Lecture Notes in Artificial Intelligence*, (pp. 83–94). Vienna, Austria: Springer Verlag.

Helft, N. (1987). Inductive generalisation: A logical framework. In *Procs. of the 2nd European Working Session on Learning.*

Kanellakis, P. (1990). Elements of relational database theory. In J. van Leeuwen (Ed.), *Formal models and semantics, handbook of theoretical computer science.* Amsterdam: Elsevier.

Kietz, J.-U. (1988). Incremental and reversible acquisition of taxonomies. In J. M. Boose (Ed.), *Proceedings of EKAW–88*, (pp. 1–11). Sankt Augustin: GMD. Also as KIT-Report 66, Technical University Berlin.

Kietz, J.-U. (1996). *Induktive Analyse relationaler Daten.* PhD thesis, Technische Universität Berlin.

Kietz, J.-U., & Džeroski, S. (1994). Inductive logic programming and learnability. *SIGART–Bulletin, 5*, 22–32.

Kietz, J.-U., & Wrobel, S. (1992). Controlling the complexity of learning in logic through syntactic and task–oriented models. In S. Muggleton (Ed.), *Inductive logic programming.* London: Academic Press.

Lavrač, N., & Džeroski, S. (1994). *Inductive logic programming — techniques and applications.* New York: Ellis Horwood.

Lindner, G., & Morik, K. (1995). Coupling a relational learning lagorithm with a database system. In Y. Kodratoff, G. Nakhaeizadeh, & C. Taylor (Eds.), *Statistics, Machine Learning, and Knowledge Discovery in Databases*, MLnet Familiarization Workshops, (pp. 163–168). MLnet.

Mannila, H. (1995). Aspects of data mining. In Y. Kodratoff, G. Nakhaeizadeh, & C. Taylor (Eds.), *Statistics, Machine Learning, and Knowledge Discovery in Databases*, MLnet Familiarization Workshops, (pp. 1–6). MLnet.

Mannila, H., & Räihä, K. (1994). Algorithms for inferring functional dependencies from relations. *Data and Knowledge Engineering, 12*, 83–99.

Mannila, H., & Toivonen, H. (1996). On an algorithm for finding all interesting sentences. In R. Trappl (Ed.), *Cybernetics and Systems '96 (EMCSR 1996)*, (pp. 973–978). Vienna: Austrian Society for Cybernetic Studies.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 1). Palo Alto, CA: Morgan Kaufmann.

Michalski, R. S. (1994). Inferential theory of learning: Developing foundations for multistrategy learning. In R. S. Michalski & G. Tecuci (Eds.), *Machine learning: A multistrategy approach* (Vol. 4). San Francisco, CA: Morgan Kaufmann.

Morik, K., Wrobel, S., Kietz, J.-U., & Emde, W. (1993). *Knowledge acquisition and machine learning: Theory, methods, and applications.* London: Academic Press.

Muggleton, S. (1995). Inverse entailment and progol. *New Generation Computing, 13*, 245–286.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19/20*, 629–679.

Pazzani, M. J. (1995). An iterative improvement approach for the discretization of numeric attributes in Bayesian classifiers. In U. M. Fayyad & R. Uthurusamy (Eds.), *Proceedings of the First International Conference on Knowledge Discovery and Data Mining* (pp. 228–233). Cambridge, MA: AAAI Press/MIT Press.

Pazzani, M. J., & Kibler, D. (1992). The utility of knowledge in inductive learning. *Machine Learning, 9*, 57–94.

Piatetsky-Shapiro, G. (1991). Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro & W. Frawley (Eds.), *Knowledge discovery in databases.* Cambridge, MA: AAAI Press/MIT Press.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Savnik, I., & Flach, P. A. (1993). Bottom-up induction of functional dependencies from relations. In G. Piatetsky-Shapiro (Ed.), *Proceedings of the AAAI-93 Workshop on Knowledge Discovery in Databases*, (pp. 174–185). Menlo Park, CA: AAAI Press.

Ullman, J. D. (1988). *Principles of database and knowledge–base systems* (Vol. 1). Rockville, MD: Computer Science Press.

Wettscherek, D., & Dietterich, T. G.(1995). An experimental comparison of the nearest-neighbour and nearest-hyperrectangle algorithms. *Machine Learning, 19*, 5–27.

Wrobel, S., Wettscherek, D., Sommer, E., & Emde, W. (1996). Extensibility in data mining systems. In E. Simoudis & J. W. Han (Eds.), *2nd Int. Conference on Knowledge Discovery and Data Mining*, (pp. 214–219). Menlo Park, CA: AAAI Press.

Received Date
Accepted Date
Final Manuscript Date