Machine Learning, 26, 147–176 (1997)

First Order Regression

ARAM KARALIČ aram.karalic@ijs.si
Jožef Stefan Institute, Ljubljana, Slovenia

IVAN BRATKO ivan.bratko@fri.uni-lj.si
Faculty of Electrical Engineering and Computer Science, University of Ljubljana, and
Jožef Stefan Institute, Ljubljana, Slovenia

Editors: Stephen Muggleton and David Page

Abstract. We present a new approach, called First Order Regression (FOR), to handling numerical information in Inductive Logic Programming (ILP). FOR is a combination of ILP and numerical regression. First-order logic descriptions are induced to carve out those subspaces that are amenable to numerical regression among real-valued variables. The program FORS is an implementation of this idea, where numerical regression is focused on a distinguished continuous argument of the target predicate. We show that this can be viewed as a generalisation of the usual ILP problem. Applications of FORS on several real-world data sets are described: the prediction of mutagenicity of chemicals, the modelling of liquid dynamics in a surge tank, predicting the roughness in steel grinding, finite element mesh design, and operator's skill reconstruction in electric discharge machining. A comparison of FORS' performance with previous results in these domains indicates that FORS is an effective tool for ILP applications that involve numerical data.

Keywords: machine learning, inductive logic programming, regression, real-valued variables, first-order logic, applications of machine learning.

1. Introduction

2. Integrating ILP and Numerical Regression

In analysing applications of ILP it has often been observed (e.g. (Bratko & Džeroski, 1995)), that one of the major practical impediments is lack of facilities for handling numerical information in currently available ILP systems. Therefore, several attempts have been made recently to extend ILP techniques with numerical capabilities (Camacho, 1994, Mizoguchi & Ohwada, 1995, Sebag & Rouveirol, 1995, Srinivasan, 1996) In this paper we present a new approach to handling numerical information in ILP, called *First Order Regression*. We define First Order Regression (FOR) as a combination of Inductive Logic Programming, using first order logic, and numerical regression. The approach has been implemented in the program FORS (First Order Regression System, (Karalič, 1995a, Karalič, 1995b)).

In the paper we present the basic ideas and algorithms of FORS, and several applications to real-world learning problems for which FOR seems to be a natural approach. These applications include the prediction of mutagenicity of chemicals, the modelling of liquid dynamics in a surge tank, predicting the roughness in steel grinding, finite element mesh design, and operator's skill reconstruction in electric discharge machining. Most of these problems have already been tackled with other ILP techniques (mutagenicity (Srinivasan *et al.*, 1994), finite element mesh design (Dolšak *et al.*, 1994), regression tree based induction

(steel grinding (Filipič *et al.*, 1991)), and equation discovery techniques (liquid dynamics (Križman *et al.*, 1995)). This enables a comparison of FORS' performance with previous results. The electric discharge machining application has not been attempted previously.

3. Integrating ILP and Numerical Regression

3.1. Hypothesis Representation in FOR

In the usual definition, inductive learning is viewed as search for formulas that discriminate between positive and negative examples, or between the possible classes. Accordingly, we are interested in inducing formulas that are true for subsets of examples that contain no negative examples (or possibly just a few, in the case of noise). We say that such a formula *covers* positive examples only.

First Order Regression (FOR) can be viewed as an extension of this in the following sense. Some arguments of the target predicate are real-valued. Such arguments will be called *continuous attributes*. FOR aims at covering subsets of examples that are amenable to numerical regression between the continuous arguments. Consequently, the result of FOR is a partition, by logical rules, of the example space into subspaces that can be handled by numerical regression.

Notice the following features of this approach:

1. There are no negative examples in FOR. This is convenient for applications in automated modelling of (possibly dynamic) systems from data. The learning data is obtained from measurements and thus only contains "positive" examples. On the other hand, the usual ILP techniques require some, possibly unnatural, way of generating negative examples (typically based on some kind of closed-world assumption).

2. The goodness of an induced rule in FOR corresponds to the degree of numerical fit between the data covered by the rule and the corresponding regression formula. So the natural measure of goodness of a candidate clause is the numerical error of the regression formula on the data covered by the clause.

We now turn to some details of how First Order Regression is carried out in FORS. FORS performs *linear* regression. However, new continuous variables can be introduced by background knowledge literals. So numerical relations are essentially not limited to linear equations. There is always a distinguished continuous attribute which by convention appears as the first argument of the target predicate. This distinguished attribute then always appears on the left hand side of regression equations, and some or all of the remaining continuous attributes, or new continuous variables, appear in the regression expressions on the right hand sides of the equations. So the induced clauses about the target predicate in FORS have the form:

```
f( Y, X1, X2, ...)  :-
   Literal1, Literal2, ...,
   Y is a1 * Xi1 + a2 * Xi2 + ..., !.
```

Here `Y` is the distinguished continuous attribute, and `Xij` are some other continuous variables. Notice that this convention in FORS does affect the generality of the class of relations inducible by FORS, although the regression literal above could be rewritten so that some other continuous attribute would appear on the left hand side of the equation. The particular form of induced clauses and regression equations assumes that the distinguished continuous attribute is a function of other variables. Also the measure of fit is taken relative to `Y` (as defined later).

In the described setting, a part of the relation describing the volume of the ideal gas can be represented as:

```
state(Vol,Gas,Pressure,Mass,Temp) :-
  Gas = hydrogen,
  mult( A, Mass, Temp), % A = Mass * Temp
  div( B, A, Pressure), % B = Mass * Temp / Pressure
  Vol is 4160.0 * B.    % Vol = 4160.0 * Mass * Temp / Pressure
state(Vol,Gas,Pressure,Mass,Temp) :-
  Gas = oxygen,
  mult( A, Mass, Temp),
  div( B, A, Pressure),
  Vol is 260.0 * B.
```

The usual induction in ILP could be emulated by FORS roughly as follows. The target predicate in usual ILP would be modified by adding an extra "continuous" attribute whose value would be determined by the truth of the examples: 1.0 for positive examples, and 0.0 for negative examples. The regression formulas would only involve this attribute. Consequently, FORS would tend to find rules that cover subsets of positive-only examples and negative-only examples.

3.2. Related Work

Quinlan and Cameron-Jones (Quinlan & Cameron-Jones, 1993) report on useful, although rather basic capability of FOIL to handle, to some degree, real-valued variables. FOIL includes literals like $V_i > k$ and $V_i > V_j$ that allow an existing numerical variable V_i to be compared against a threshold k found by FOIL, or against another variable V_j of the same type.

Camacho presented an ILP system Indlog (Camacho, 1994) that includes mechanisms for coping with a large number of examples, that include noisy and numerical data. The system learns from positive and negative examples. Indlog was applied to the problem of cloning pilot's flying skills with some good preliminary results.

LINUS and DINUS (Lavrač & Džeroski, 1994, Džeroski, 1995) map an ILP problem to a propositional learning problem. So the induction is delegated to an attribute-based learner. Employing regression tree induction program RETIS (Karalič, 1992), LINUS and DINUS can efficiently handle numerical and noisy data. They are however limited to essentially propositional hypothesis language.

Mizoguchi and Ohwada's program (Mizoguchi & Ohwada, 1995) produces a constraint logic program from a set of problem solutions. They define constraint-based subsumption (C-subsumption), and use it in the construction of a hypothesis as the *least general constraint set*. The idea is implemented in the system GKS and applied to the problem of spatial layout. GKS also learns from positive and negative examples.

Sebag and Rouveirol's approach (Sebag & Rouveirol, 1995) is based on an idea of transforming a maximally discriminant generalisation problem into an equivalent constraint satisfaction problem. An induced hypothesis results from solving such a constraint satisfaction problem.

Srinivasan's experiments in numerical reasoning in ILP (Srinivasan, 1996) involve numerical inequalities, linear regression and differential equations. Of all the related work, this is probably the most similar to FORS, but Srinivasan's approach still requires both positive and negative examples which is often unnatural in real-world domains. Induction of differential equations in FORS, as well as in Srinivasan's work, was inspired by the LAGRANGE program (Džeroski & Todorovski, 1995).

4. Hypothesis Language of FORS

Here we give a precise definition of FORS' hypothesis language and corresponding declarations. A hypothesis induced by FORS is a Prolog program. Background knowledge consists of standard Prolog predicate procedures. The induced part of the hypothesis consists of clauses of the form:

`f(Y,V1,V2,...,Vn) :- literal1, ... literaln,!.`

All clause literals are positive.

By analogy with attribute-value learning, variable `Y` is called *class variable* while variables V_1 to V_n are *attributes*. Literals can be one of the following:

- a call of a background knowledge predicate,
- A *variable value literal* — a literal of the form `Variable = value` for discrete variables and of the form `Variable =< const` or `Variable >= const` for real-valued variables, used to test values of real-valued and discrete variables,
- a recursive call $f(Y', V'_1, V'_2, \ldots, V'_n)$,
- a regression literal of the form `Y is Expression`, where `Expression` is either a number or a linear combination of real-valued variables.

FORS' input consists of the following parts:

- Declaration of types of variables that will be used. There are three kinds of variable types: discrete, real-valued, and term. The value of a term variable can be any Prolog term.
- Declaration of variables.

- Learning examples. Examples are ground facts about the target predicate. Notice that there are no negative examples.

- Specification of background knowledge. Background knowledge consists of arbitrary Prolog predicates. The specification for one background knowledge predicate consists of its Prolog procedure and its declaration. The declaration of a predicate contains the information about the types of the arguments and their input/output modes and also indicates whether a literal calling the predicate should be treated as a normal, determinate or total literal. Determinate and total literals are defined below.

A *determinate* literal is a literal for which the values of output variables are uniquely determined for every instantiation of input variables. We define a *total* literal as a determinate literal that succeeds for every call with instantiated input and uninstantiated output variables.

FORS uses information about totality of literals when it decides whether to add a default clause at the end of the induced hypothesis. If the last clause in the hypothesis induced so far consists solely of total literals, then it will always succeed and no additional (default) clause needs to be added.

The above is illustrated by an example of learning a program for computing the factorial of a natural number. Figure 1 depicts the domain definition, the background knowledge, and the learning examples for this problem. FORS was presented the factorials of the first five natural numbers. As background knowledge, procedures for multiplication and decrement were used. Because they always succeed, they were declared as total. With enabled learning of recursive definitions, FORS induced a correct definition of $n!$, also shown in Figure 1.

Input for FORS:

```
% Target predicate
target( f( -integer, +integer)).
% Examples
f(1,0). f(1,1). f(2,2). f(6,3). f(24,4).
% Background knowledge
bkl( decr( -integer, +integer), total).
bkl( times( -integer, +integer, +integer), total).
decr( N1, N) :- N1 is N - 1.
times( R, A, B) :- R is A * B.
```

Result, produced by FORS:

```
f(F,N) :- N =< 1, F is 1,!.
f(F,N) :- decr(A,N), f(B,A), times(F,N,B),!.
```

Figure 1. Example of hypothesis language and background knowledge specification of FORS.

5. The Mechanisms of FORS

5.1. Algorithms

FORS uses the covering approach similar to the one used in FOIL (Quinlan, 1990). The algorithm repeatedly constructs clauses. When a clause is found, all examples covered by the clause are removed from the learning set. The procedure is repeated while there are enough examples left (parameter MinExs). Finally, a default clause is added if necessary. The default clause is a clause with no condition part (i.e., a clause which always succeeds and assigns a value to the class variable). It is added if the previous clauses of the hypothesis do not guarantee instantiation of the class variable.

During learning FORS uses values of several user-definable parameters, described in Figure 2.

Figure A.1 (in Appendix A) depicts the top-level covering algorithm of FORS. The interpretation of induced clauses is limited by a maximal call depth. Examples that can not be proved within this depth, are considered uncovered. The clause building part uses a top-down approach. The algorithm starts with the most general candidate clause, covering the entire example set (i.e., clause with empty condition part) and then specializes it by adding literals. Clause construction uses beam search through the space of possible clauses. This part of the algorithm is presented in Figure A.2.

Single specialization step of a clause is depicted in Figure A.3. First, an attempt is made to specialize the clause with background knowledge literals, then with variable-value literals and finally, with recursive literals. All specializations can be enabled and disabled through user-definable parameters.

After each clause is constructed, redundant literals are removed. No other more sophisticated post-processing of the induced model is done.

5.2. Error Estimate

For every candidate clause an error estimate is calculated aimed at predicting the clause's error when classifying unseen examples. The clause error on the covered learning example is used as an error estimate. The *mean squared error* (R) is used as an error estimate in our case. It is defined as

$$R(\mathcal{C}, \mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} (y_e - f_{\mathcal{C}}(e))^2, \tag{1}$$

where $\mathcal{E}$ denotes the example set covered by clause $\mathcal{C}$, y_e denotes the value of the example's class and $f_{\mathcal{C}}(e)$ denotes the value of the example's class as predicted by the clause $\mathcal{C}$. *Relative mean squared error* RE of hypothesis $\mathcal{H}$ is defined as

$$RE(\mathcal{H}, \mathcal{E}) = \frac{R(\mathcal{H}, \mathcal{E})}{R(\mathcal{H}_\mu, \mathcal{E})}, \tag{2}$$

Parameter	Description	Default value
MinExs	Minimal number of examples that an induced clause must cover.	10
MinErrRed	Minimal error reduction.	1 %
AllErr	Allowed error.	1 %
MaxVarDepth	Maximal allowed variable depth.	2
MaxLits	Maximal number of literals in a clause.	30
MaxStar	Maximal number of clauses in a star.	5
MaxRegVars	Maximal number of variables in linear regression term. Value 0 indicates that linear regression should not be attempted.	0
DetTotThresh	Threshold for applying determinate and total literals (see Figure A.3). Value 0.0 indicates that determinate and total literals should not be applied.	80 %
UseDiscAVLits	Flag used to indicate whether to use literals of the form `Attribute = value` for discrete variables.	`yes`
UseContAVLits	Flag used to indicate whether to use literals of the form `Attribute >= const` and `Attribute =< const` for real-valued variables.	`yes`
LearnRecursive	Flag used to indicate whether FORS should try to learn recursive definition.	`no`
UseMDL	Flag indicating whether to use MDL heuristics to prune the search space.	`no`
DiffWin	Differentiating window half-width. When differentiating real-valued variable of example i, examples $e_{i-\mathsf{DiffWin}}$ to $e_{i+\mathsf{DiffWin}}$ are used for differentiation.	5

Figure 2. FORS' parameters.

where $\mathcal{H}_\mu$ represents a hypothesis which always gives mean class value of the entire learning example set as a predicted class value. RE is usually between 0 and 1, although a very poor hypothesis can produce RE greater than 1. Relative mean squared error enables comparison of hypotheses from different domains. It is never used during the construction of a hypothesis, but it is useful in evaluating induced hypothesis on test data.

5.3. Stopping Criteria

The criteria for stopping clause refinements in FORS are based on:

- minimal number of examples that a clause must cover,
- maximal variable depth,
- maximal number of literals in a clause,
- minimal improvement of candidate clause with respect to its predecessor,
- error of the clause is within allowed error AllErr,
- minimum description length (MDL) principle.

Explanation of the last three criteria follows.

Minimal Local Improvement When the clause is specialized with one literal, its error R changes. Minimal error improvement MinErrRed can be specified by the user. The new clause is considered a suitable candidate clause only if improvement in R exceeds the specified minimal improvement. The only point at which this criterion is not considered is when FORS adds determinate and total literals, since primary goal of determinate and total literals is the introduction of new variables and not the improvement of clause's error.

Allowed Error When dealing with real-valued variables, especially when their values come from measurements, it is naive to expect a hypothesis that perfectly fits the data. We are usually satisfied with hypotheses that have sufficiently small error AllErr. When a clause with an error smaller than AllErr is encountered, it is considered as acceptable and the search is stopped.

MDL Pruning Another noise-fighting mechanism is the MDL-pruning. It is an application of Rissanen's minimal description length (MDL) principle (Rissanen, 1978), which has become a common technique for estimating the point at which a theory starts overfitting the data, thus fitting the noise present in the data (Quinlan & Rivest, 1989, Muggleton *et al.*, 1992, Kovačič, 1995).

The MDL principle states that the best theory to explain the data is the one which minimizes the sum of the length (in bits) of the theory description, and the length (in bits) of the data when coded with the help of the theory (Li & Vitanyi, 1993).

A hypothesis (clause) is considered *compressive* if the sum of the coding lengths of the clause and the additional information needed to precisely reconstruct the covered examples, using the clause, is smaller than the sum of coding lengths of examples' class values. When the MDL pruning is applied, only compressive hypotheses are considered promising, while uncompressive ones are discarded.

To use MDL pruning in FORS, the MDL principle implementation had to be devised which can also handle real-valued variables. Details about the implementation can be found in (Karalič, 1996, Karalič, 1995a).

5.4. Linear Regression

To allow for detection of linear relationships between the real-valued attributes and the class, FORS has a facility to construct a regression plane through the class values of the covered examples. It is thus possible to add a literal of the form e.g.

```
Y is 1.0 * A + 2.7 * B,                                    (3)
```

which could better grasp the relationship between the attributes and the class. However, during the clause construction, many new variables could be introduced. The linear regression procedure would therefore often find itself in a position to construct a n-dimensional hyperplane through less than n data points, which would lead to underdetermined hypothesis. To reduce the number of such situations, a user-definable parameter MaxRegVars is set by the user to determine maximal number of terms in linear regression formula. It is expected that the user ensures MinExs $\geq$ MaxRegVars.

5.5. Time Series Handling

Differential equations are a common way to describe dynamic processes in time. Differential equations typically describe the relationship between the observed quantities and their time derivatives. However, not all time derivatives are normally available as measurements. Therefore, a system capable of modelling time series should incorporate means for computing time derivatives of observed quantities.

When FORS is faced with a time series, each example should represent the state of variables at one point in time. Intervals between successive time points need not be equal. FORS can derive any variable with respect to the time variable.

A numerical differentiation method using linear regression is used. Successive time points are approximated by a straight line and its derivative is used as a derivative estimate for the central point. The width of the window is determined by parameter DiffWin. Approximation of a line for example e_k is determined by examples $e_{k-\text{DiffWin}}$ to $e_{k+\text{DiffWin}}$.

The reason that this method is preferred to methods which draw a polynomial *through* all the points is that this technique also acts as a simple high-pass filter (Hamming, 1989). The technique was already successfully employed to automated modelling of dynamic systems from measurements in (Korenjak, 1994, Križman, 1993).

5.6. Recursive Definitions

The usage of recursive literals is controlled by the parameter LearnRecursive. When LearnRecursive is true, recursive literals are added as follows. All combinations of input

variables (obeying type declarations) are considered, and the output variable (that is: class variable) is treated as a new variable. A literal is added if the resulting clause covers at least MinExs examples. Infinite recursion during evaluation of recursive clauses is prevented by FORS' limitation on call-depth.

5.7. Implementation

FORS is currently implemented in C and SICStus Prolog 2.1 #4 and runs on Sun workstations. The main part of the system consists of approximately 6000 lines of C. A simple depth-bound Prolog interpreter is written in SICStus Prolog and called from C to interpret induced clauses.

6. Applications in Real-World Domains

6.1. Mutagenicity of Nitroaromatic Compounds

6.1.1. Introduction

The problem of discovering rules for mutagenicity in nitroaromatic compounds comes from the field of organic chemistry. Mutagenic activity has often been found to be linked to carcinogenesis and damage to DNA. Chemists typically use graph-based constructs such as atom/bond connectivities or Cartesian coordinates of atoms to describe molecules. Srinivasan et al. (Srinivasan *et al.*, 1994, Srinivasan & Muggleton, 1995, Srinivasan *et al.*, 1995) were the first to apply ILP program Progol to the mutagenicity problem with impressive results.

6.1.2. Data

We have used data about 188 chemical compounds, studied in (Srinivasan *et al.*, 1995). Mutagenicity is a real number, so numerical regresion was naturally used for prediction. However, to enable comparison with previous work in this domain (Srinivasan *et al.*, 1995), we measured FORS's accuracy in terms of binary predictions: discriminating compounds with positive log mutagenicity from those which have zero or negative log mutagenicity, as done in (Srinivasan *et al.*, 1995).

Background knowledge consisted of two parts: primitive structural representation of molecules and generic structural knowledge. Primitive structural representation is comprised of Prolog facts of the form:

```
bond(Compound,Atom1,Atom2,Bondtype)
```
and

```
atm(Compound,Atom,Element,Atomtype,Charge).
```

Table 1. Classification accuracy of FORS hypotheses in the mutagenicity domain.

UseMDL	MinExs	MaxRegVars 0	1	2
yes	1	77.7± 9.4	83.0± 9.4	83.5±11.0
no	100	66.5±12.2	67.0±11.3	76.1± 7.1
no	50	78.2±12.7	72.9±15.0	84.0± 8.7
no	20	81.9±10.3	81.9±10.4	87.8± 9.8
no	10	86.2± 9.1	84.0± 8.7	87.2± 6.4
no	8	88.8± 5.5	85.6± 9.6	82.5± 6.2
no	4	85.1± 6.9	87.8± 6.3	85.6± 6.2
no	2	87.8± 5.7	86.2± 9.4	87.2±12.3
no	1	89.4± 5.7	84.0± 6.6	83.5± 6.0

The data were originally obtained from the standard molecular modelling package QUANTA (Srinivasan *et al.*, 1995). There are 10136 ground facts about `bond/4` and `atm/5` for the 188 molecules.

Generic structural knowledge defines elementary chemical concepts, such as methyl groups, hetero-aromatic rings, and the three distinct topological ways to connect three benzene rings.

Additionally, two real-valued attributes, describing important chemical properties, were utilized: a measure of hydrophobicity $\log P$ and the energy of the compound's lowest unoccupied molecular orbital ϵ_{LUMO}.

6.1.3. Experiments

Ten-fold cross validation was used with exactly the same partitioning of the data as in (Srinivasan *et al.*, 1995) of roughly same sized learning/testing example sets. The results of ten tests were then averaged.

Learning was performed with default parameter settings, except for three parameters (MinExs, UseMDL, and MaxRegVars), whose settings were varied as follows:

- Use MDL pruning, minimal number of examples MinExs = 1.
- Do not use MDL pruning, minimal number of examples MinExs = 100, 50, 20, 10, 8, 4, 2, 1.
- Maximal number of linear regression variables MaxRegVars = 0, 1, 2.

Table 1 presents average classification accuracy and standard deviation for the various settings. Notice the accuracy 86.2 ± 9.1 for the default settings of parameters (UseMDL =no, MinExs = 10, MaxRegVars = 0).

Comparison with the performance of other algorithms, studied in (Srinivasan *et al.*, 1995) is given in Table 2. In the Table, NS refers to two non-structural attributes $\log P$ and ϵ_{LUMO}, PS refers to additional predefined structural attributes, which were used in the systems

Table 2. Comparison of FORS with other systems in the mutagenicity domain. The results of other algorithms are taken from (Srinivasan *et al.*, 1995)

Algorithm	Accuracy
Linear regression + NS	85 ± 3
Linear regression + NS + PS	89 ± 2
Neural network + NS	86 ± 3
Neural network + NS + PS	89 ± 2
CART + NS	82 ± 3
CART + NS + PS	88 ± 2
Progol + NS + S	88 ± 2
FORS + NS + S	89 ± 6
FORS + NS + S + MDL	84 ± 11
Default class	66 ± 3

that can not utilize structural background knowledge, S refers to structural knowledge, and MDL refers to MDL pruning.

The results by FORS are compared with the results obtained with Progol, linear regression model, back-propagation neural network, and a version of CART. Srinivasan et al. (Srinivasan *et al.*, 1995) give more details about these other algorithms. For linear regression, neural network and CART, two experiments were performed: one with only the two non-structural (NS) attributes $\log P$ and ϵ_{LUMO}, used also in our study, and the other with additional structural attributes (SA), which were used in the experiments since those algorithms can not utilize structural background knowledge. We can see that FORS performed comparably to the other algorithms.

As an illustration, Figure 3 presents one of the best models generated during the experiments. The hypothesis correctly classified 100 % of testing examples, and was constructed in 13.7 minutes (on Sun SPARCstation 10) using nondefault parameter values MinExs = 20, MaxVarDepth = 1, and MaxRegVars = 2.

6.1.4. Conclusions

In this section the problem of predicting mutagenic activity of nitroaromatic compounds was addressed. The problem is suitable for first-order algorithms, since the compounds are described using topological relations between their atoms. Additionally, two non-structural attributes were used, which describe important chemical properties of the compounds. The results obtained with FORS were compared with results of some other approaches: Progol, linear regression model, back-propagation neural network, and a version of CART. The results show that FORS compares well with the other systems.

It can be seen in Table 2 that MDL technique does indicate some appropriate level of pruning, although some fine tuning of other parameters can lead to better accuracy.

```
f(LogAct,Compound,Logp,Lumo) :- % This clause covers 25 examples.
    Logp =< 1.8600,
    LogAct is -2.44432688 - 1.09152865 * Lumo,!.
f(LogAct,Compound,Logp,Lumo) :- % This clause covers 23 examples.
    phenanthrene(Compound,A),
    atm(Compound,B,C,D,E),
    D >= 22.0,
    LogAct is  - 0.89506185 * Lumo - 15.20483971 * E,!.
f(LogAct,Compound,Logp,Lumo) :- % This clause covers 27 examples.
    ring_size_5(Compound,A),
    Lumo >= -1.6900,
    LogAct is -3.83296180 - 4.21773863 * Lumo,!.
f(LogAct,Compound,Logp,Lumo) :- % This clause covers 22 examples.
    Lumo >= -1.2700,
    LogAct is -4.99677467 + 1.33741307 * Logp,!.
f(LogAct,Compound,Logp,Lumo) :- % This clause covers 27 examples.
    benzene(Compound,A),
    Logp =< 3.0000,
    LogAct is -6.24283123 - 3.19510269 * Lumo,!.
f(LogAct,Compound,Logp,Lumo) :- % This clause covers 21 examples.
    atm(Compound,A,B,C,D),
    Logp =< 4.1900,
    C >= 22.0,
    LogAct is  - 1.84037900 * Lumo - 0.08746605 * C,!.
f(LogAct,Compound,Logp,Lumo) :- % This clause covers 24 examples.
    atm(Compound,A,B,C,D),
    D >= -0.1100,
    LogAct is  - 1.23747396 * Lumo + 20.39890862 * D,!.
f(LogAct,Compound,Logp,Lumo) :- % Default clause, covers 169 examples.
    LogAct is -2.38454413 - 2.04081726 * Lumo,!.
```

Figure 3. A hypothesis induced in the mutagenicity domain.

It should be noted that FORS, unlike Progol, actually predicts the *numerical* value of mutagenicity, and not only "active" or "inactive". Only to enable comparison, FORS' results were reduced to binary.

6.2. Finite Element Mesh Design

6.2.1. Introduction

The finite element method (FEM) is a frequently used engineering tool for determining stresses in physical structures. In order to apply FEM, a structure must be divided into a finite set of elements. Deformations of elements in the structure are then computed using linear algebraic equations. The CPU time needed to solve the set of equations depends on the number of elements comprised in the structure. The number of elements also affects the numerical accuracy of the computed solution. The accuracy increases with the number of

elements, but so does the CPU time. There is no known numerical method for determining a suitable partition into finite elements of a given object. Considerable expertise, based on experience, is required for choosing appropriate number of elements, because one has to achieve sufficient accuracy of the computed solution and at the same time maintain CPU time at a reasonable level. In (Dolšak & Muggleton, 1992) and (Dolšak *et al.*, 1994), ILP has been applied to learn general rules about how to suitably partition a given object. In our experiment with FORS we used Dolšak's data (Dolšak *et al.*, 1994).

6.2.2. Experiments

In our experiment, the data describing five objects, labelled A, B, C, D, and E, were used. Appropriate number of mesh elements for each edge in the objects was determined by an expert and verified computationally. The target relation was

```
f(N,Edge)
```

The relation defines the number of elements `N` for each edge `Edge`. Background knowledge consisted of relations describing the properties of an edge (e.g. `short(E)`, `not_loaded(E)`), boundary conditions (e.g. `free(E)`), loadings (e.g. `not_loaded(E)`), and the relations describing the structure of the object (e.g. `neighbour(E1,E2)`). Detailed description of the objects, as well as some previous work in the domain can be found in (Dolšak *et al.*, 1994, Dolšak & Muggleton, 1992).

In the experiments, four objects were used for learning and the remaining object was used for testing. This was repeated five times, for each possible choice of the objects for learning and testing. The best results were produced with the following non-default parameter values:

- minimal number of examples MinExs = 2,
- maximal variable depth MaxVarDepth = 1,
- maximal number of literals in a clause MaxLits = 4.

As an illustration, Figure 4 shows the theory constructed from objects B, C, D, E, and used to classify edges of the object A. The induction took 51 minutes of CPU time on Sun SPARCstation 10.

Usage of MDL pruning produced theories with slightly, but not significantly, lower accuracy. Table 3 compares achieved results with the results of the FOIL (Quinlan, 1990), mFOIL (Džeroski, 1991), GOLEM (Muggleton & Feng, 1990), MILP (Kovačič, 1995), and FFOIL (Quinlan, 1996). Results for FOIL and FFOIL were obtained from (Quinlan, 1996), results for GOLEM and MILP were obtained from (Kovačič, 1995).

FORS performed better than FOIL and mFOIL, roughly similar to GOLEM and MILP, and worse than FFOIL.

```
f(N,Edge) :- quarter_circuit(Edge),N is 9,!.
f(N,Edge) :- circuit_hole(Edge),N is 8,!.
f(N,Edge) :- not_important(Edge),N is 1,!.
f(N,Edge) :- short_for_hole(Edge),N is 1,!.
f(N,Edge) :- short(Edge),N is 2,!.
f(N,Edge) :- one_side_loaded(Edge),N is 12,!.
f(N,Edge) :- one_side_fixed(Edge),N is 2,!.
f(N,Edge) :- long_for_hole(Edge),N is 6,!.
f(N,Edge) :- half_circuit(Edge),cont_loaded(Edge),N is 6,!.
f(N,Edge) :- circuit(Edge),cont_loaded(Edge), N is 8,!.
f(N,Edge) :- circuit(Edge),opposite(Edge,A), N is 8,!.
f(N,Edge) :- circuit(Edge),N is 12,!.
f(N,Edge) :- long(Edge),fixed(Edge), N is 7,!.
f(N,Edge) :- half_circuit(Edge),opposite(Edge,A),half_circuit(A),N is 6,!.
f(N,Edge) :- half_circuit_hole(Edge),not_loaded(Edge),opposite(Edge,A),N is 6,!.
f(N,Edge) :- half_circuit(Edge),neighbour(Edge,A),usual(A),N is 8,!.
f(N,Edge) :- half_circuit(Edge),N is 6,!.
f(N,Edge) :- half_circuit_hole(Edge),neighbour(Edge,A),long_for_hole(A),N is 6,!.
f(N,Edge) :- half_circuit_hole(Edge),not_loaded(Edge),N is 4,!.
f(N,Edge) :- long(Edge),N is 11,!.
f(N,Edge) :- free(Edge),opposite(Edge,A),equal(Edge,B),N is 2,!.
f(N,Edge) :- half_circuit_hole(Edge),N is 7,!.
f(N,Edge) :- neighbour(Edge,A),quarter_circuit(A),N is 2,!.
f(N,Edge) :- neighbour(Edge,A),long(A),N is 3,!.
f(N,Edge) :- neighbour(Edge,A),short_for_hole(A),N is 2,!.
f(N,Edge) :- neighbour(Edge,A),one_side_loaded(A),N is 3,!.
f(N,Edge) :- equal(Edge,A),usual(A),N is 4,!.
f(N,Edge) :- neighbour(Edge,A),free(A),fixed(Edge),N is 4,!.
f(N,Edge) :- neighbour(Edge,A),circuit(A),N is 2,!.
f(N,Edge) :- neighbour(Edge,A),half_circuit(A),not_loaded(Edge),N is 7,!.
f(N,Edge) :- neighbour(Edge,A),usual(A),fixed(A),opposite(Edge,B),N is 3,!.
f(N,Edge) :- neighbour(Edge,A),not_important(A),N is 6,!.
f(N,Edge) :- neighbour(Edge,A),opposite(Edge,B),short(A),N is 2,!.
f(N,Edge) :- neighbour(Edge,A),short(A),one_side_fixed(A),N is 5,!.
f(N,Edge) :- neighbour(Edge,A),usual(A),neighbour(Edge,B),short(B),N is 3,!.
f(N,Edge) :- neighbour(Edge,A),usual(A),N is 3,!.
f(N,Edge) :- N is 4,!.
```

Figure 4. Rules for finite element mesh design.

6.2.3. Conclusions

The achieved results are comparable with the results of other algorithms in this domain. However, the overall accuracy is still pretty low and can easily be achieved by naive Bayesian classifier using only background knowledge relations of arity 1. It should be noted, however, that the simple accuracy is not the most natural measure of success in this domain and does not reflect well the actual applicability. Also, expert interpretation of the results of learning (Dolšak, 1996) favours relational representation as opposed to mere attribute representation.

Table 3. Number and percentage of correctly classified edges in FEM design.

structure	FOIL	mFOIL Laplace	mFOIL $m = 0$	GOLEM	MILP	FFOIL	FORS
A	16	23	22	21	21	21	22
B	9	12	12	12	12	15	12
C	8	9	9	10	11	11	8
D	12	6	6	16	16	22	16
E	16	12	12	21	30	54	29
$\sum$	36	62	61	80	90	123	87
%	22	22	22	29	32	44	31

6.3. Surge Tank

A surge tank is a container that is usually connected to the pressure pipes that conduct water to the turbines of a water hydro-power plant. If a sudden change of the flow through the turbines occurs, a pressure surge is generated in the pipeline. The surge can have serious consequences on the pipeline (explosion or implosion), so a surge tank is situated as close as possible to the place of formation of the surge. Surge pressure is transformed to the water movement in the surge tank, resulting in an increase or decrease of the steady-state water level in the tank. The rest of the pipeline is thus not exposed to the pressure shocks (Križman *et al.*, 1995).

6.3.1. Experiments

Our experiments were conducted with measurement data of water movement in a laboratory replica of a surge tank. Only one variable was measured: the water level h. The data consists of time series describing the water level. The level is measured in equal time intervals $dt = 0.496$ seconds at 126 time points, which yields 126 training examples. In the experiments variable h was differentiated twice, yielding two new variables $\dot{h}$ and $\ddot{h}$. 2 proved to be the best window width for differentiation.

Hypotheses which cover the whole example set with just one differential equation fit the examples very poorly. Therefore in further experiments minimal number of examples that a clause must cover was set sufficiently low allowing induction of at least two clauses. The best induced hypothesis is:

```
f(H_dot_dot,H_dot,H) :-
    H_dot =< 0.0030,
    times(A,H_dot,H_dot),
    H_dot_dot is  - 0.05831576 * H + 3.51532865 * A,!.
f(H_dot_dot,H_dot,H) :-
    times(A,H_dot,H_dot),
    H_dot_dot is  - 0.05301992 * H - 3.84900069 * A,!.
```

and can be rewritten as:

$$\ddot{h} = \begin{cases} -0.05831576\ h + 3.51532865\ \dot{h}^2 & \text{if } \dot{h} \leq 0.003 \\ -0.05301992\ h - 3.84900069\ \dot{h}^2 & \text{otherwise} \end{cases}.$$

6.3.2. Expert's Evaluation and Conclusions

The best theoretically derived model of this tank is considered by experts to be

$$\ddot{h} = -0.05608\ h - 3.75771\ \dot{h}|\dot{h}|.$$

We will now evaluate the model induced by FORS and compare its performance with the models induced by GOLDHORN (Križman, 1993) and with the performance of the best known model (the "correct" model). Performance of the "correct" model will also be assessed because even this model does not reproduce the measured system's behavior perfectly. In the rest of the section, t_i denotes time at the i-th point; $h_m(t)$ denotes measured water level at time t; while $h_c(t)$, $h_g(t)$, $h_f(t)$ denote water height at time t according to the "correct", GOLDHORN's, and FORS' model, respectively.

As a measure of a model's correctness, the difference between the *behavior* according to the model, and the measured behavior is taken. A domain expert (Kompare, 1995) suggested using ${}_a\Delta_b = \sum_{i=1}^{N} |h_a(t_i) - h_b(t_i)|$ as a difference measure ${}_a\Delta_b$ between two behaviors a and b.

Table 4. The difference between several tank's behaviors.

"correct"Δmeasured	=	0.2905
FORSΔmeasured	=	0.3110
GOLDHORNΔmeasured	=	0.3610

Table 4 gives the quality of various models. We see that the "correct" behavior is better than the other two behaviors and that FORS' behavior is slightly better than the behavior produced by GOLDHORN, but all the models produce roughly the same error compared to the measurements.

According to the domain expert, the most important criterion of model quality is ${}_.\Delta_.$, so he regarded our induced model as a success. The expert also commented that learning the model without prior knowledge of the "absolute value" function (its usage was necessary for GOLDHORN) can be considered as an important achievement.

6.4. Steel Grinding

The task in steel grinding domain is to determine the roughness of the workpiece from the properties of the sound produced during the process of steel grinding (Junkar, *et al.*, 1991). The broader aim of the study was to elaborate the results from the point of view of machine control, where the task is to perform relevant control actions in response to parameter values

monitored during the process execution. An example of a control action is terminating the process when its performance falls below an unacceptable degree (Filipič *et al.*, 1991). Since control action is easily deducible from workpiece roughness, the subproblem of workpiece roughness estimation was addressed first. Several machine learning techniques have already been applied to this problem, yielding encouraging results and showing that machine learning tools can produce adequate models of roughness prediction (see (Filipič *et al.*, 1991) or review paper (Rowe *et al.*, 1994)).

6.4.1. Experiments

Measurement data were obtained from an experiment in which vibration signals generated by the grinding wheel and the workpiece were measured by an accelerometer sensor and processed by a spectrum analyser. >From the obtained spectra, predefined spectral features were extracted: *total spectrum area* (`SpArea`), *frequency of the maximum area peak* (`MaxAreaX`), and *frequency of the spectrum area central point* (`AreaCX`). Simultaneously, workpiece surface roughness was measured.

Two background knowledge literals were defined: `=<` and `>=`, enabling FORS to compare the frequencies. An experiment was then carried out to evaluate the contribution of so defined background knowledge compared to results obtained previously with attribute-based techniques (Filipič *et al.*, 1991). Several learning sessions were performed with different pre-pruning mechanisms and results of each session were thoroughly evaluated by a domain expert.

6.4.2. Expert's Evaluation and Conclusions

It turned out that the use of background knowledge did not bring any significant improvement in the model quality in terms of predictive accuracy. However, the newly induced models frequently contained background knowledge literals. It is interesting that according to domain experts (Posel, 1995), this is a significant improvement, because newly induced models are more general than the models without background knowledge. This is because they do not rely so much on absolute numerical values for specific attribute value thresholds, but instead they evaluate attribute values relatively. For example, without using background knowledge, a literal such as `MaxAreaX <= 6125` typically appeared, saying that the frequency of the maximum area peak is less then 6125 Hz. In this particular setting (grinding regime, choice of the tools) this means that the maximum area peak is in the lower part of the spectrum. Using different grinding wheel speed the whole spectrum would shift to higher or lower frequencies, thus making the above literal useless, because it now means something completely different. However, usage of background knowledge typically yielded literal `MaxAreaX <= AreaCX`, which directly states that the maximum peak must lie in the lower part of the spectrum, regardless of the frequency area in which the spectrum is situated, therefore enabling rule usage in much broader class of working regimes.

The domain expert generally preferred models using linear regression with at most one variable, because they revealed more information than the previous models. Experiments

with two or three variables were also conducted, but they were harder to comprehend while not revealing significant new details.

As an illustration, we present a model, constructed using MDL pruning in Figure 5.

```
f(Roughness,SpArea,MaxAreaX,AreaCX) :-
                    AreaCX =< 5600, MaxAreaX <= AreaCX, SpArea =< 0.25,
                    Roughness is 1.70,!.
f(Roughness,...) :- MaxAreaX <= AreaCX, AreaCX =< 5890, MaxAreaX =< 1140,
                    Roughness is 0.00169 * MaxAreaX,!.
f(Roughness,...) :- AreaCX >= 7310, SpArea >= 0.2130,
                    Roughness is 0.00016832 * AreaCX,!.
f(Roughness,...) :- MaxAreaX <= AreaCX, SpArea >= 0.33, AreaCX >= 5950,
                    Roughness is 0.00029069 * AreaCX,!.
f(Roughness,...) :- SpArea =< 0.1700, Roughness is 0.98,!.
f(Roughness,...) :- AreaCX =< 5890, Roughness is 1.72,!.
f(Roughness,...) :- SpArea =< 0.23, Roughness is 5.41015863 * SpArea,!.
f(Roughness,...) :- SpArea >= 0.31, AreaCX >= 6190,Roughness is 1.38,!.
f(Roughness,...) :- MaxAreaX >= 5730, SpArea >= 0.33,
                    Roughness is 4.13661194 * SpArea,!.
f(Roughness,...) :- AreaCX <= MaxAreaX, Roughness is 1.28,!.
f(Roughness,...) :- Roughness is 1.82,!.
```

Figure 5. A model of the steel grinding process.

Domain expert claims to be satisfied with the results of machine learning, since our models enabled him to grasp some additional process properties which he wouldn't be able to discover only with classical statistical tools. However, he suggested that the machine learning approach should not be used alone but should be considered as a powerful supplement to the already existent instruments. It is also worth noting, that if the expert had to choose between two equally good models in terms of relative error, he usually chose the larger model, justifying the decision by stating, that the smaller model did not grasp sufficient detail. However, when choosing amongst the models which had 1, 2 or 3 variables in linear regression terms, he chose a model with only one variable in linear regression terms, claiming that it was easier to understand. He also preferred such models over the models with no linear regression.

6.5. Electrical Discharge Machining

In *electrical discharge machining* (EDM), the workpiece surface is machined by electrical discharges occurring in the gap between two electrodes — the tool and the workpiece. The gap is continuously flushed by the third element, the dielectricum (Junkar *et al.*, 1993). The process consists of numerous randomly ignited monodischarges generating crater-textured surface. Stability and quality of the process depend on various parameters: type of dielectricum, workpiece material, required surface roughness, size of the gap between the electrodes (that is, between the electrode and the workpiece), servomechanism strengthening, period of discharges, duration of discharges, electric current upper limit, open voltage.

Some parameters can be controlled during the process (gap, flow), some require process interruption (duration of discharges, ...), and some of them are inherent to the particular machining task and can not be changed (e.g., required workpiece roughness). For a standard set of workpiece types there exist predefined (recommended by the manufacturer of an EDM machine) sets of values for the process parameters, which guarantee certain degree of generated surface quality. However, the parameter settings are very conservative and do not yield really good performance in terms of the time needed to accomplish the task. Therefore, a human operator is normally employed to control the process parameters and minimize the machining time.

One immediately sees the potential importance of automatically reproducing operator's behavior: an EDM machine assisted by an "automatic" operator. So in this domain the goal of induction was not to model the physics of a dynamic system, but to reconstruct an operator's skill from examples.

6.5.1. Experiments

During the conversations with the operator, it was found that he monitors the following characteristics: (i) Relative share of different kind of pulses generated during dischargement. (There are three kinds of pulses: `A` (*empty*) pulses, `B` (*effective*) pulses, and `C` (*arc*) pulses.) (ii) average electric current, (iii) gap, and (iv) flow. The operator controls only gap and flow. The values of other quantities remained fixed through all experiments. Mean values and deviations of the observed quantities for the last 5 and the last 20 seconds were chosen as the attributes. The name of an attribute consists of three letters. The first letter indicates the parameter (`A`, `B`, `C` for the three kinds of pulses and `I` for the electric current). The second letter describes time interval: `L` is used for attributes describing state of the process in the last 20 seconds (`L`ong-term) while `S` (for `S`hort-term) is used to describe the last 5 seconds. The third letter is `M` for the `M`ean value or `D` for the standard `D`eviation.

It is interesting that during the process of attribute identification the operator frequently monitored some quantities which he claimed he monitored only occasionally or that he didn't monitor at all. This showed up when the operator was observed during one of the test runs.

To enable the detection of the rate of change of the attributes, the predicate "`<`" was defined as background knowledge. The attributes were given appropriate types, so that the comparison of their values (through background knowledge) was possible only for the attributes of the same type.

Since we tried to model control by two variables (gap and flow), the learning task was decomposed in two learning tasks — learning gap control and learning flow control separately.

For each control variable three actions were possible: increase the control variable (the action was assigned a numerical value +1.0), no action (value 0.0), and decrease the control variable (value -1.0).

For each action (gap or flow change) a learning example was generated *in each domain*. If, for example, gap was increased, one example was generated for the "gap" (sub)domain, with action +1.0 and one example was also generated for the "flow" (sub)domain, with

action 0.0. Additionally, to provide learning examples also in the areas of relatively stable process behavior, for each 60 seconds elapsed without an action, an example was generated with action 0.0 for both subproblems.

6.5.2. Expert's Evaluation and Conclusions

Several models were generated, using various levels of pre-pruning. As an example of a generated skill model, a pair of submodels is presented here:

```
f(DGap ,ASM,ASD,BSM,BSD,CSM,CSD,ISM,ISD,ALM,ALD,BLM,BLD,CLM,CLD,ILM,ILD) :-
                ILD >= 0.55, DGap is 0.3,!.
f(DGap ,...) :- CLM >= 5.6300, DGap is 0.7,!.
f(DGap ,...) :- BSM =< 1.1000, DGap is -0.2,!.
f(DGap ,...) :- DGap is -0.5,!.

f(DFlow,...) :- ASD =< 0.0100, DFlow is 0.0,!.
f(DFlow,...) :- ALD =< 0.0800, DFlow is 0.0,!.
f(DFlow,...) :- ASM < ALM, ISD < ILD, BSD < BLD, DFlow is 0.1,!.
f(DFlow,...) :- BLM < BSM, DFlow is 0.5,!.
f(DFlow,...) :- DFlow is 0.1,!.
```

`DGap` means *change of gap*, `DFlow` means *change of flow*. Notice that the induced model may instantiate `DGap` and `DFlow` to values *between* -1.0 and 1.0 although in the learning examples these values were only -1.0, 0.0, and 1.0. In an automatic controller, the rules could be interpreted probabilistically, determining the *proportion* of the corresponding directions of change of gap and flow. The operator's and expert's comments on the model (documented in (Junkar & Komel, 1996) and (Komel, 1996)) were that most of the induced actions were logical. However, there were some comments about the understandability of the models:

- In operator's opinion the decomposition of the problem (separate control rules for gap and flow) degraded model's understandability. A combination of gap and flow actions would be more intuitive.

- Clauses which appear late in the hierarchy of clauses tend to be less comprehensible, because one must remember the negations of all the clauses that appeared before.

- There were also situations where simple post-processing of the clause bodies (in combination with a solution to previous remark) would improve their comprehensibility. For example, in the situation

  ```
  f(...) :- ALM =< 0.10,...,ALM =< 0.05
  ```

  the first literal is obviously redundant. In fact, domain expert usually rewrote the rules prior to their evaluation.

To improve model comprehensibility, the two induced submodels were combined together to reveal the combined strategy for simultaneous control of gap and flow. The combination of the models was done in the following way:

1. Every clause in the model was expanded, so that clauses explicitly included negations of the conditional parts of previous clauses' bodies. This step ensured that each model consisted of mutually exclusive clauses with unordered evaluation order.

2. Each clause in the gap model was combined with each clause of the flow model to produce new set of rules for every possible combination of conditional parts of clauses' bodies.

The above procedure resulted in a model consisting of 20 disjoint clauses, each predicting gap *and* flow actions. As an illustration, we present the clause constructed from the last clause of the gap submodel and the last clause of the flow submodel:

```
f(DG/DF,...):-
    ILD  < 0.55, CLM  < 5.63, BSM  > 1.10, ASD  > 0.01,
    ALD  > 0.08, ASM >= ALM,  ISD >= ILD,  BSD >= BLD, BLM > BSM,
    DG is -0.5, DF is 0.1,!.
```

Despite the fact that the new model contained more than two times as many clauses as the original and that all the clauses were larger, the operator and the expert claimed that this model is more comprehensible than the first one.

The domain expert qualitatively defined the relation between the effectiveness of the process and the control parameters — gap and flow. The qualitative states of the process are presented in two dimensions on the left diagram of Figure 6. The diagram defines two states of the process: stable state and arcing state. Effectiveness of the process decreases as the distance of the process from the boundary between the two regions increases. The arcing state is to be avoided, because it can damage the workpiece. Stable process region, on the other hand, is not harmful to the workpiece. It is however not recommendable to stay in the stable process region, because its characteristic is very low process effectiveness. So, the goal of the control should be to guide the process as close as possible to the boundary between the arcing and the stable region. More precisely: even on the boundary the performance is not constant. There is a (small) region on the boundary, which represents the region of the best process performance. The region is marked with an asterisk. The state diagram was further divided into six regions, representing qualitatively different process behaviors.

The induced model was evaluated by plotting the suggested actions into the state diagram of the process. For each rule in the model, at least one vector representing the combination of suggested gap and flow actions was drawn, representing the suggested action. For some rules, more than one vector was drawn, because if the rule was very general, it could cover more than one region of the state diagram. Right side of the Figure 6 depicts actions suggested by the induced model.

One can immediately see that most of the vectors point towards the boundary between the regions — the optimal working region. Additionally we see that many of them point towards the point of optimal performance. It seems that the model correctly reproduces

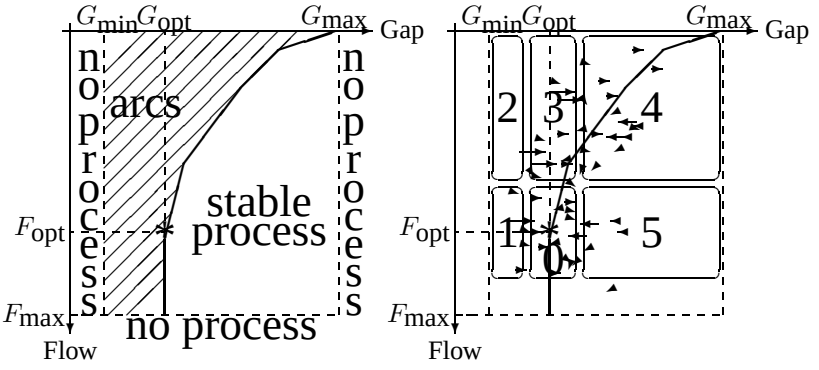

Figure 6. State space of the EDM process and control actions predicted by the induced model.

the operator's skill. The model has been installed on the actual machine to successfully replace the human operator. Expert evaluation together with the comparison of some other approaches to this problem can be found in (Komel, 1996).

7. Discussion

We presented a new approach, called First Order Regression (FOR), to handling numerical information in Inductive Logic programming. In FOR, first-order logic descriptions are induced to carve out those subspaces that are amenable to numerical regression among the continuous variables. The program FORS is an implementation of this idea, where numerical regression is focussed on distinguished continuous argument of the target predicate, called class. It is assumed that class is a function of other arguments. This can be viewed as a generalisation of the usual ILP problem as follows. In usual ILP, the task is to induce a definition of the target predicate, that is a function to $true, false$. On the other hand, FORS induces definitions of functions to real numbers, and the language for defining these functions is a combination of first-order logic and numerical regression equations.

Other features of FORS are: the use of background knowledge in intensional form, enable modelling of dynamic systems by inducing differential equations from time series data, handling of noisy data. FORS incorporates several mechanisms to handle noise, including MDL pruning and allowed maximal regression error. The MDL pruning was generalised to the handling of continuous variables.

7.1. Achieved Results

The aim of FORS was to alleviate the lack of numerical facilities in ILP, which was often felt by the users of ILP programs on real world data. Experiments with FORS in a number of real world domains show that FORS is an effective tool in this respect. Some of the results and observations can be summarised as follows. Regarding prediction accuracy, FORS' performance compared well with the published results obtained by other systems. In the mutagenicity application, hypotheses induced by FORS give the actual numerical prediction which is the natural type of prediction in this domain. In earlier experiments with other ILP systems, prediction was reduced to binary (active or inactive) due to lack of numerical capabilities. The modelling of water oscillations in a surge tank proved that FORS can successfully handle time series data. In steel grinding, the use of background knowledge enabled by FORS did not improve prediction accuracy in comparison with the previous regression tree induction. However, as a result of background knowledge, the models induced by FORS are more general and enable better expert interpretation. This was considered a significant improvement by the domain expert. In electrical discharge machining, the induced model of the operator's skill has been successfully installed as a controller on the actual machine as a replacement for the human operator.

7.2. Space and Time Complexity

Although we have not done an in-depth theoretical analysis of the FORS algorithm, the following observations are important from the practical point of view.

Space complexity of FORS didn't present any problems. However, time complexity often turned out to be most critical for practical use. Sometimes we were able to get a result in reasonable time (up to an hour) only by restrictive parameter settings. The most critical factors were the size of the background knowledge, maximal number of literals in a clause, and maximal variable depth. Execution times are stated for each discussed model. Execution times vary from less than one second (for synthetic domains), to several hours on Sun SPARCstation 10. Typically, in the applications in real domains described in the paper, CPU time was several minutes to half an hour. However, we usually interrupted sessions which took more than one hour of CPU time and tried to achieve speed-up with different parameter settings.

7.3. The Role of MDL Pruning

FORS has quite a few user-definable parameters. This allows flexible use; it enables the user to tailor, through successive experiments, the induction process to the properties of the application domain. This is in accord with general experience of ML applications which encourages domain exploration by repeated learning experiments whereby varying the parameters of the system and modifying the representation.

It may seem difficult for the user to navigate through the rather large space of parameter settings. But here the MDL pruning helps by adopting the following policy for parameter

tuning. First, run FORS with MDL pruning and a "loose" setting of the other parameters (e.g. MinExs = 1), which basically leaves the control of pruning to the MDL mechanism. The result of this gives a general idea of FORS' performance in a new domain. Further experiments typically vary the minimal number of parameters that a clause must cover (MinExs) which enables fine tuning. Turning the use of attribute-value literals off when they are not expected in the model, significantly speeds up the induction process. The same holds for total literals. In some domains (mesh design, mutagenicity) it was necessary to restrict the number of literals in a clause to keep induction times reasonable. The overall impression is that it is better to start with more restrictive parameter settings, observe the algorithm's behaviour, and then gradually allow FORS to search larger and larger hypothesis spaces.

Regarding the MDL pruning, it seems that its major role is in helping the user to develop an initial "feeling" for a new domain. Therefore MDL is typically used in initial stages of experimentation. Later, when the user has some basic idea of appropriate parameter settings, MDL pruning is switched off. Then small further improvements are often possible by fine tuning of other parameters.

7.4. FORS *and Propositional Learners*

It could be argued that all the constructed models could also be obtained using some of the well-known regression-tree algorithms or other ILP systems. However, there is to our knowledge no single system applicable to all the domains discussed in this paper. In the following paragraphs we analyse advantages of FORS with respect to propositional learners and other ILP systems with numerical facilities.

Of course in quite a few cases the models could be learned by classical regression trees algorithm, but only if one knew *in advance* which attributes to use and "manually" derive from relational background knowledge. As usual in ILP, the power of FOR is that we define all potentially useful relations and then let the algorithm construct new variables (attributes). If we wanted to achieve the same results with a propositional learner, we would have to define a large number of attributes which would often make this ineffective and impractical.

The results in the mesh domain nicely illustrate that relational representation is crucial. Namely, in the models induced in that domain, a substantial amount of clauses appear which exploit characteristics of neighbouring edges. For example, the clause

```
f(N,Edge) :-
   neighbour(Edge,A),
   usual(A),
   neighbour(Edge,B),
   short(B),
   N is 3,!.
```

applies to edges which have one "usual" neighbouring edge and one short neighbouring edge. To induce such rule using propositional systems one would have to, prior to induction, define a variety of attributes describing whether an edge has a usual neighbour, whether an edge has a short neighbour etc. With FOR, as usual in ILP, the same effect is achieved by

defining a single relation `neighbour/2`. It is also important to note that in the opinion of a FEM expert (T. Hellen, documented in (Dolšak, 1996)), the relational description better captures knowledge used by experts when designing FE meshes.

7.5. FORS *and Other Numerical Learning in ILP*

We see two major advantages of our approach with respect to other ILP systems with numerical capabilities.

- FORS incorporates several features, available in many different systems. For example, in the surge tank domain, only the availability of various features in a single system enabled us to achieve the best results in this domain so far. The combination of essential features includes:
  - use of background knowledge,
  - proper treatment of continuous variables,
  - handling of time series,
  - partitioning the attribute space, and
  - use of linear regression.
- FOR *does not require negative examples*. This is convenient for applications in automated modelling of (possibly dynamic) systems from data. The learning data is obtained from measurements and thus only contains "positive" examples. On the other hand, the usual ILP techniques require some, possibly unnatural, way of generating negative examples (typically based on some kind of closed-world assumption).

Appendix A

Appendix A contains pseudo-code description of the FORS algorithm. The algorithm is discussed in Section 5.1. The notation used in the descriptions of FORS' algorithms is as follows:

$\mathcal{H}$	= Induced hypothesis.
$\mathcal{C}$	= A clause.
$\mathcal{C}.R$	= Clause's error estimate.
$\mathcal{H}.\mathcal{C}$	= Clause $\mathcal{C}$ as a part of the hypothesis $\mathcal{H}$.
$\mathcal{H}.\mathcal{C}_{\text{last}}$	= The last clause in the hypothesis.
$\mathcal{E}$	= Example set.
$\mathcal{S}$	= A star, i.e. a set of candidate clauses in beam-search.
$\lvert\mathcal{S}\rvert$	= Number of clauses in a star.
$\mathcal{S}.\mathcal{C}_i$	= i-th clause of a star.

A star consists of candidate clauses waiting to be further specialized. Clauses are ordered by their error estimate — the first clause having the lowest error.

During learning FORS uses values of several user-definable parameters. To enable easier reading of the algorithm pseudo-code description, Figure 2 gives a description of the parameters.

```
1.   procedure InduceHypothesis(var H)
2.       begin
3.           H:= ∅; E:= "All learning examples"
4.           MakeDefaultClause(E,C0)
5.           while not StopLearning(E) do begin
6.               GenerateClause(C,E)
7.               AddClauseToHypothesis(C,H)
8.               RemoveExamplesCoveredByClause(C,E)
9.           end
10.          if (H=∅) or not HasOnlyTotalLiterals(H.Clast)
11.          then AddClauseToHypothesis(C0,H)
12.      end
```

Figure A.1. Top level of the FORS algorithm.

```
1.   procedure GenerateClause(var C,E)
2.   { C = Generated clause. E = Example set. }
3.       begin
4.           InitClause(C,E)
5.           S:= ∅; S.C1 := C; Cbest := C
6.           while (Cbest.R>AllErr) and (|S| >0) do begin
7.               Sold := S; S:= ∅
8.               for i:=1 to |S| do BeamStepForOneClause(Sold.Ci,S,Cbest)
9.           end
10.          C:= Cbest
11.      end
```

Figure A.2. Top level of clause generation.

```
1.  procedure BeamStepForOneClause(C,var S,var C_best)
2.  { One step of clause specialization. }
3.  { C = clause to be specialized. C_best = best clause so far. }
4.      procedure Spec2(var MaxImp)
5.      { MaxImp = maximal achieved improvement of clause's error. }
6.          begin
7.              if UseBKLits then SpecClWithBKnLiterals(C,S,C_best)
8.              SpecClWithVarValLiterals(C,S,C_best)
9.              if LearnRecursive then SpecClWithRecLiterals(C,S,C_best)
10.         end
11.     begin
12.         if C.NoLiterals<MaxLits then begin
13.             Spec2(MaxImp)
14.             if MaxImp<DetTotThresh then begin
15.                 n_L = C.NoLiterals
16.                 SpecClWithDetAndTotLiterals(C,S,C_best)
17.                 if C.NoLiterals> n_L then Spec2(MaxImp)
18.             end
19.         end
20.     end
```

Figure A.3. One step of clause specialization.

Acknowledgments

We thank R.D. King, S.H. Muggleton, A. Srinivasan, and M.J.E. Sternberg for the data used in the mutagenicity experiments, B. Dolšak for the mesh data, B. Kompare for the surge-tank data, M. Junkar for the steel grinding data, and I. Komel for the electrical discharge machining data.

B. Dolšak and B. Kompare helped as domain experts for mesh design and surge tank respectively, M. Junkar and R. Posel, helped as domain experts for steel grinding, I. Komel and V. Kralj helped as domain experts for electrical discharge machining.

The work described in this paper was supported by the Slovenian Ministry of Science and Technology (project Knowledge Synthesis from Data) and Commission of the European Communities (project ILP).

References

Bratko, I., & Džeroski, S. (1995). Engineering applications of ILP. *New Generation Computing*, 13, 313–333.

Camacho, R. (1994). Learning stage transition rules with Indlog. *Proceedings of the Fourth International Workshop on Inductive Logic Programming (ILP-94)* (pp. 273–290), Bad Honnef/Bonn, Germany: Gesellschaft für Mathematik und Datenverarbeitung Sankt Augustin.

Dolšak, B., Bratko, I., & Jezernik, A. (1994). Finite element mesh design: An engineering domain for ILP application. *Proceedings of the Fourth International Workshop on Inductive Logic Programming (ILP-94)* (pp. 305–320), Bad Honnef/Bonn, Germany. Gesellschaft für Mathematik und Datenverarbeitung Sankt Augustin.

Dolšak, B. (1996). *A Contribution to Mesh Design for Finite Element Method*. PhD thesis, University of Maribor, Faculty of Mechanical Engineering, Maribor, Slovenia.

Dolšak, B., Jezernik, A., & Bratko, I. (1994). A knowledge base for finite element mesh design, *Artifical Intelligence in Engineering*, 9, 19–27.

Dolšak, B., & Muggleton, S. (1992). The application of inductive logic programming to finite-element mesh design. Stephen Muggleton, editor, *Inductive Logic Programming*: Academic Press.

Džeroski, S. (1991). *Handling noise in inductive logic programming*. Master's thesis, University of Ljubljana, Faculty for Electrical Engineering and Computer Science, Ljubljana, Slovenia.

Džeroski, S. (1995). *Numerical Constraints and Learnability in Inductive Logic Programming*, PhD thesis, University of Ljubljana, Faculty for Electrical Engineering and Computer Science, Ljubljana, Slovenia.

Džeroski, S., & Todorovski, L. (1995). Discovering dynamics: from inductive logic programming to machine discovery. *Journal of Intelligent Information Systems*, 4, 89–108.

Filipič, B., Junkar, M., Bratko, I., & Karalič, A. (1991). An application of machine learning to a metal-working process. *Proceedings of ITI-91*, 167–172, Cavtat, Croatia.

Hamming, R. W. (1989). *Digital Filters*: Prentice-Hall.

Junkar, M., Filipič, B., & Bratko, I. (1991). Identifying the Grinding Process by Means of Inductive Machine Learning. *Computers in Industry*, 17(2–3), 147–153.

Junkar, M., Filipič, B., & Žnidaršič, M. (1993). An AI approach to the selection of dielectricum in electrical discharge machining. *Proceedings of Third International Conference on Advanced Manufacturing Systems and Technology AMST'93*, Udine, Italy.

Junkar, M., & Komel, I. (1996). Knowledge Acquisition for Adaptive Control of the EDM Process. *Proceedings of 15th IAESTED International Conference on Modelling, Identification and Control* (pp. 295–297), Innsbruck, Austria.

Karalič, A. (1992). Employing linear regression in regression tree leaves. *Proceedings of ECAI'92 (European Conference on Artificial Intelligence)* (pp. 440–441), Vienna, Austria.

Karalič, A. (1995a). *First Order Regression*. PhD thesis, University of Ljubljana, Faculty for Electrical Engineering and Computer Science.

Karalič, A. (1995b). First order regression: Application in real-world domains. *Proceedings of Artificial Intelligence Techniques — AIT'95*, Brno, Czech Republic.

Karalič, A. (1996). Producing More Comprehensible Models While Retaining Their Performance. *Proceedings of Information, Statistics and Induction in Science ISIS'96* (pp. 54–65), Melbourne, Australia.

Komel, I. (1996). *Surface Modelling and Expert Knowledge Acquisition for the Control of electrical Discharge Machining Process*. Master's Thesis, University of Ljubljana, Faculty of Mechanical Engineering.

Kompare, B. (1995). Faculty of Civil Engineering and Geodesy, Department for Hydroengineering, Institute for Sanitary Engineering. Ljubljana, Slovenia. Personal communication.

Korenjak, B. (1994). *Experimental Environment for Modelling of Dynamic Systems Using Artificial Intelligence Methods*. Master's thesis, University of Ljubljana, Faculty for Electrical Engineering and Computer Science, Ljubljana, Slovenia. In Slovene.

Kovačič, M. (1995). *Stochastic Inductive Logic Programming*. PhD thesis, Faculty of electrical Engineering and Computer Science, Ljubljana, Slovenia.

Križman, V. (1993). *Handling Noisy Data in Automatic Modelling of Dynamical Systems*. Master's thesis, University of Ljubljana, Faculty for Electrical Engineering and Computer Science, Ljubljana, Slovenia. In Slovene.

Križman, V., Džeroski, S., & Kompare, B. (1995). Discovering dynamics from measured data. *Working Notes of the MLNet Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases*. Institute of Computer Science, Heraklion, Greece.

Lavrač, N., & Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chicester.

Li, M., & Vitányi, P. (1993). *An Introduction to Kolmogorov Complexity and its Applications*. Texts and Monographs in Computer Science. Springer-Verlag.

Mizoguchi, F., & Ohwada, H. (1995). An inductive logic programming approach to constraint acquisition for constraint-based problem solving. *Proceedings of the 5th International Workshop on Inductive Logic Programming (ILP-95)* (pp. 297–322): Katholieke Universiteit Leuven, Heverlee, Belgium.

Muggleton, S., & Feng, C. (1990). Efficient induction of logic programs. *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, Japan.

Muggleton, S., Srinivasan, A., Bain., M. (1992). Compression, significance and accuracy. *Proceedings of Machine Learning Conference 1992* (pp. 338–347), Aberdeen.

Posel, R. (1995). Expert System for Roughness Prediction During Steel Grinding. *Proceedings of Management of Inovative Technologies - MIT'95* (pp. 296–303), Bled, Slovenia.

Quinlan, R. (1990). Learning Logical Definitions From Relations. *Machine Learning*, 3(5).

Quinlan, R. (1996), University of Sydney, Sydney, Australia, Personal communication.

Quinlan, R., & Cameron-Jones, M. (1993). FOIL: A Midterm Report. *Proceedings of Sixth European Conference on Machine Learning*, (pp. 3–20), Vienna, Austria.

Quinlan, R., & Rivest., R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation*, 80, 227–248.

Rissanen, J. (1978). Modelling by shortest data description. *Automatica*, 14, 465–471.

Rowe, W. B., Yan, L., Inasaki, I., & Malkin, S. (1994). Applications of Artificial Intelligence in Grinding (Keynote paper). *Annals of the CIRP*, 43(2), 521–531.

Sebag, M., & Rouveirol, C. (1995). Constraint inductive logic programming. *Proceedings of the 5th International Workshop on Inductive Logic Programming (ILP-95)* (pp. 297–322): Katholieke Universiteit Leuven, Heverlee, Belgium.

Srinivasan, A. (1996). Experiments in numerical reasoning with ILP. Michie, D., Muggleton, S., & Furukawa, K., editors, *Machine Intelligence 15*. Oxford University Press.

Srinivasan, A., & Muggleton, S. H. (1995). Comparing the use of background knowledge by two inductive logic programming systems. *Proceedings of the 5th International Workshop on Inductive Logic Programming (ILP-95)* (pp. 199–230): Katholieke Universiteit Leuven, Heverlee, Belgium.

Srinivasan, A., Muggleton, S. H., King, R. D., & Sternberg, M. J. E. (1994). Mutagenesis: ILP experiments in a non-determinate biological domain. *Proceedings of the Fourth International Workshop on Inductive Logic Programming (ILP-94)*, (pp. 217–232): Bad Honnef/Bonn, Germany. Gesellschaft für Mathematik und Datenverarbeitung Sankt Augustin.

Srinivasan, A., Stephen, S. H., Sternberg, M. J. E., & King, R. D. (1995). Theories for mutagenicity: a study in first-order and feature-based induction. Technical Report PRG-TR-8-95, Oxford University Computing Laboratory, Oxford.

Received December 1995
Accepted July 1996
Final Manuscript December 1996