Implementing Numerical Reasoning in ILP

Demšar, Damjan[1] and Gams, Matjaž[1]

Abstract - We introduce the 4S system capable of handling numerical problems within the general Inductive Logic Programming (ILP) framework. Previous systems had severe problems with handling numerical domains, as well as with using background knowledge and multi-relational data at the same time. The idea introduced in 4S is that regression results are integrated into an ILP program as numerical formulas. A formula is created dynamically according to the available examples and hypotheses. In this way, 4S can deal with numerical problems presented in multi-relational data by using background knowledge. The logic-based and numeric-based techniques in 4S collide in the presence of complex recursion, but we believe that the functional power of our system is less limited than related ones. We analyze the performance of 4S.

Inductive logic programming, Linear regression, Machine learning

1. INTRODUCTION

Various attempts have been made to implement Inductive Logic Programming (ILP) systems that are capable of handling numerical data [4], [22], [29], [30]. The general idea is to combine ILP with numerical manipulation in order to enlarge the space of problem domains that can be successfully tackled by ILP systems. Basic ILP has several advantages [21], such as the ability to deal with data stored in multiple relational tables and to take into account background (domain) knowledge expressed as a logic program. ILP also uses powerful expression language to describe logical relations.

Pure ILP, however, is not capable of handling numerical data [2], especially when the output "class variable" is continuous. ILP systems usually do not differentiate input and output variables, while numerical problems often involve differentiation. By itself, ILP cannot even perform such numerical functions as (in)equalities, arithmetic, trigonometry, geometry and linear regression. Of these, regression is the most relevant one in machine-learning problems since numerous real-life models can be approximated by formulas. Regression enables construction of a formula from a given set of examples, but it lacks logical mechanisms for constructing more advanced logical models on top of formulas. ILP and regression are therefore ideal for combining different advantages into an advanced integrated system, which we refer to as first order regression [14].

FORS [14] (First Order Regression System) was one of the first implemented ILP systems capable of handling numerical data. First Order Regression is an integration of (first order logic) Inductive Logic Programming and numerical regression. After first experiments with the implemented FORS system, we started with an advanced algorithm and system 4S.

The paper is organized as follows: In Section 2 we describe definitions of ILP, regression and integration. In Sections 3 and 4 we describe the 4S [6] system as an algorithm and as a fully operational program. In Section 6 we present tests on artificial and real-world problems.

2. ILP, REGRESSION AND FIRST ORDER REGRESSION

For our purposes, ILP can be defined in the following way:

- **B** is background knowledge consisting of a set of definite clauses
- E is a set of examples $E = E^+ \cup E^-$ where:
- $E^+ = \{e_1, e_2, \ldots\}$ are positive examples that are definite clauses, often ground unit clauses.
- $E^- = \{\overline{f_1}, \overline{f_2}, \ldots\}$ are negative examples that are Horn clauses, often ground unit clauses.
- $B \not\models E^+$, where $X \models Y$ denotes semantic entailment of Y by X
- Given B and E, the output is a set of definite clauses $H = \{D_1, D_2, \ldots\}$, that is a complete and consistent explanation of the examples, from a predefined language *L*. H usually satisfies at least the following conditions:
  - Each D_i explains at least one positive example. That is $B \wedge D_i \models e_j \vee e_k \vee \ldots$, where $e_m \in E^+$
  - $B \wedge H \models E^+$
  - $B \wedge H \not\models f$, $f \in E^-$

Linear regression is defined as follows:

- E is a set of examples in the form $y_n = f(x_{1,n}, x_{2,n}, \ldots, x_{m,n})$, where $x_{i,j}$ are real-valued

[1] Jožef Stefan Institute, Ljubljana, Slovenia
{damjan.demsar, matjaz.gams}@ijs.si

parameters.

- Given E, the output is a set of parameters A_0, A_1, …, A_m, that minimizes the sum of squares $\Sigma_n(y_n-(A_0+A_1x_{1,n}+A_2x_{2,n}+\dots+A_mx_{m,n}))^2$

We propose the following integration of ILP and regression:

- **B** is background knowledge consisting of a set of definite clauses
- **E** is a set of examples e_n in the form of ground unit clauses $f(y_n, x_{1,n}, x_{2,n},\dots, x_{m,n})$ where $x_{i,j}$ are either discrete, real-valued or arbitrary terms while y_j are real-valued
- Given **B** and **E**, the output is a set of definite clauses **H** = {$\mathbf{D_1}$, $\mathbf{D_2}$,… } from a predefined language ***L***, that is a good and consistent explanation of the examples. **H** satisfies at least the following conditions:
- $\mathbf{B} \wedge \mathbf{H} \models f(y'_n, x_{1,n}, x_{2,n},\dots, x_{m,n})$
- $\Sigma_n (y_n - y'_n)^2$ is "loosely" minimal
- Each $\mathbf{D_i}$ covers at least a preset number of examples.

3. 4S SYSTEM

4S is a descendant and upgrade of FORS. Some aspects of FORS and 4S are common: both systems have the same input, learning and testing examples and definitions of background knowledge predicates. They both produce a working Prolog program that describes the relation in the input data.

4S and FORS are based on a modified FOIL [27] algorithm (which is a basic covering algorithm), which was developed for searching relations between discrete classes. In order to handle continuous classes efficiently, specific modifications are needed. The main difference is lack of negative examples. We don't have any negative examples at all, so we need to redefine suitability of a hypothesis.

In ILP, a hypothesis is usually suitable when it is complete (covers all positive examples) and consistent (does not cover any negative examples). In our case we still use completeness, i.e. we usually generate hypotheses that cover the whole range of input variables values. On the other hand, since we do not have negative examples, we cannot use consistency to eliminate unsuitable hypothesis (or clauses). To eliminate unsuitable clauses, 4S uses mean squared error combined with a user defined suitability limit. The other major modification in 4S regarding classical inductive logic programming systems is based on a need to handle continuous classes. In 4S, the class variable is predicted with a linear combination of continuous variables.

The 4S algorithm is implemented as a 4s_work procedure (Figure 1), which constructs Hypothesis, i.e. H. It uses beam-search with a user-defined beam, set by default to 10.

```
H = {};
while (|E|>0) {
WorkClauses= {DefaultClause};
NewClauses = {};
repeat
OldWorkClauses=WorkClauses;
NewClauses = RefineClauses(WorkClauses);
WorkClauses=BestOf(WorkClauses∪NewClauses);
until (WorkClauses == OldWorkClauses);
D = BestClause(WorkClauses);
H = H ∪ {D};
E = E - CoveredExamples(D)
}
if (needed) H = H ∪ {FailSafeClause}
```

Figure 1. 4S_work procedure

In each step all current clauses are refined with an addition of a literal (see subsection 3.2).

After a literal is added, we calculate the coverage of the new, refined clause. From the covered examples all numerical parameters are taken (all valid combinations are generated, each representing one example for linear regression), linear regression is performed and the error is calculated. Then the output of the linear regression is transformed into a Prolog equation and added to the clause.

Each generated clause must satisfy preset conditions (e.g. the number of examples covered, variable depth…). Since these conditions are a part of algorithm parameters their explanation can be found in Section 3.4. If the error of the clause is found to be sufficiently small, refinement of the current clauses is stopped and the current clause is added to the hypothesis.

3.1. Differences to FORS

Since we had no working implementation of FORS available, the implementation of 4S, does not base on the code of FORS. but on its performance. In this subsection we list some new mechanisms that are not available in FORS.

The condition that usually stops the specialization of clauses and prevents the overfitting to the learning examples is the demand that every clause covers a minimal number of examples. However, with the addition of new literals the examples expand and split into multiple sub-examples, generated from original ones. For example when we have an example tuple (a, 2), and apply a background knowledge call of a predicate that can for that example return two possible values 10 and 15, the example tuple (a, 2) splits into 2 new example tuples (a, 2, 10) and (a, 2, 15). In this way,

examples can expand into enough sub-examples to satisfy the minimal-examples condition in each clause. In that case overfitting can occur since from only one original example several seemingly different examples can be produced. To fight this problem, we used a mechanism that treats all sub-examples that were created from the same original learning example as one example. In that case every clause must cover a certain number of original examples that were provided in the input data. 4S has therefore two ways of counting examples – one of only original examples and one of virtually created examples.

When 4S is trying to set a suitable constant for the comparison literal (Var < const) there are two possible ways of finding the value the constant should have. The first way is precise but slow. It accepts each possible value that comes into consideration. Then, specialization is performed. A new clause is generated from the old one with the addition of a comparison literal with the selected value. Linear regression is performed and average error is calculated. Only the clause with the lowest average error is accepted.

The second way is much faster but less precise. It is based on the fact that the clause under specialization already contains a linear equation E. Again each possible value for the constant is taken under consideration, and all examples covered by the original clause that satisfy the new condition. The average error is predicted using the equation E. The value with the lowest predicted average error is chosen. The specialized clause is generated, linear regression is performed, and actual error is calculated. However, it is possible that the value producing the lowest average error is not the best choice.

Also parameters that limit the depth of recursion calls and add a possibility of predicting constants and not linear equations were added.

Some of the mechanisms available in FORS were left out – for example the MDL evaluation and the usage of information whether a background predicate is determinate or not (to decide whether to add a fail safe clause or not).

3.2. Language

The hypothesis language of 4S is Prolog. A hypothesis together with the supplied background knowledge predicates is a working program in Prolog. Hypotheses generated by 4S have the following form:

```
func(Y,X1,X2,... ,Xn):-
literal1,
literal2,
...
literalm,
Y is linear_equation,
!.
```

Here Y is a class variable, X1... Xn are input data variables or attributes, and linear_equation is a linear combination of input variables and variables that we get as an output with background knowledge predicates calls.

A literal can be one of:

1. Background knowledge predicate call
2. Comparison of a continuous variable with a constant (Xi >= constant or Xi =< constant)
3. Unification of a discrete variable with a constant (Xi = constant)
4. Recursive call func(Y', X'1, X'2,... ,X'n)

3.3. Input data

The input data consists of:

1. Type declarations that can be one of three kinds: discrete, continuous (real-valued) or term (any Prolog term), e.g. type(real, continuous), where real is the name of the type.
2. Declaration of the target function/predicate, e.g. target(linear('Y','X')).
3. Variable declarations. Variables used in target predicate declaration are bound to specific types, e.g. variable('X', real).
4. Background knowledge predicates declaration includes data about types and input/output modes of the arguments, e.g. bkl(cos(-real, +real), total), where +real means that the first argument is input argument of type real, -real means that the second argument is output argument of type real.
5. Background knowledge predicates definition is actually the Prolog procedure that is used whenever background knowledge predicate is called, e.g. sin(R,A):- R is sin(A).
6. Learning data are the facts about the target predicate. Unlike classical ILP systems, 4S does not use any negative examples. An example of a learning example is lrn(linear(4.30,1)).
7. Testing data, e.g. tst(linear(4.00, 0)).
8. Parameters, e.g set(max_allowed_error, 0.001), See Table 1.

3.4. Parameters

The 4S system uses the parameters in Table 1 to guide search through a hypothesis space. By setting these parameters we adjust 4S to a particular domain. To achieve best results, we must understand the problem domain and performance of 4S. We cannot expect to do better, since it is impossible to design a uniform ILP system that can handle numerical problem

from arbitrary domains without any adjustments. Among possible solutions, setting parameters is the simplest solution. 4S can be applied with parameters set by humans or other programs.

Table 1 Parameters used in 4S

Parameter	Default	Explanation
`min_examples`	10	Minimal number of examples each clause must cover
`absulute_examples`	True	Only original examples and not subexamples count for min_examples
`max_call_depth`	10	Maximal depth of recursion
`max_literals`	10	Maximal length of a clause
`max_clauses`	10	Maximal length of a hypothesis
`max_variable_depth`	10	Maximal depth of variables
`max_allowed_error`	$1*10^{-18}$	Clause with an average error of less than that is always accepted (deemed as perfect)
`min_improvement`	0	How much specialized clause must be better from its predecessor
`allow_recursive`	False	Whether recursion is permitted
`extensive_comparing`	True	Is slower but more precise selection of comparison constant allowed
`sv_decomposition`	True	Is robust but slow SV matrix decomposition rather than fast but sensitive LU decomposition used for linear regression
`allow_regression`	True	Is regression permitted
`max_regression_vars`	-1	How many variables can be used in a linear equation (-1 means all)

4. PROBLEMS WITH INTEGRATION

The combination of recursion and linear regression causes a feedback deadlock: the linear regression process needs an instantiated result of a recursive call, whereas the recursive calls require the result of linear regression. This occurs in all functions of the form *f(n) = g(f(n-1))* where *g(m)* is a linear function.

There are a couple of ways to resolve this mutual dependency, but each introduces additional difficulties. The first way is to expand recursive calls until we reach a bound, and perform an evaluation with already instantiated returned recursive parameter. This approach is sometimes referred to as lazy or late bounding or instantiation [29]. The problem with this solution is that it demands capabilities of symbolic computing. Furthermore, there is no guarantee that this process would produce enough examples for linear regression, and no guarantee that the solution would be reasonable.

The second way to deal with the deadlock is to use the learning examples in a non-circular way. In this way, only learning examples that have known values for recursive calls (known from learning examples, or previously accepted clauses) are taken for input into the linear regression process. However, this demand is difficult to meet in real-world problems – it demands that the available learning examples cover precisely those cases that the recursive calls require. Clearly, only synthetic problem domains will have such nice properties. Since we wanted 4S to be applicable to real-world problems, we did not find this solution to be acceptable.

The first simple solution implemented in 4S is to limit the recursion and/or complexity depth. In case of too deep a recursion, the program discards the current clause and with another one.

To cope with recursion, we introduced a limited set of parameters that enable solving a specific set of predefined recursive forms. For example, solving n! can be found by limiting the linear equation to the simple form of $y = x_i$. This approach certainly cannot solve more complex cases of recursion, but can handle simple ones. In a similar way, it is possible to tune 4S so that it finds various predefined special forms. The problem with this approach is that with more complex expressions, the tuning and related hand coding gets more and more time consuming. On the other hand, we expect that a couple of simple cases suffice for expected real-life problems. In face of the complexity of the task, this seems a sensible approach.

Putting aside potential mutual deadlock between recursions and regression, other ILP mechanisms are intact in 4S. If we discard regression, 4S constructs recursive ILP programs in the same manner as other ILP systems.

5. RELATED WORK

Systems like LAGRANGE [9] and GOLDHORN [19] discover equations from numerical data, but since they are not ILP systems they lack its expressive power. There are several approaches adding numerical capabilities to ILP systems: restricting the hypothesis language to logical atoms [25], using built-in definitions for inequalities [4], [26], [27], using transformations of data to propositional level [21], using background knowledge for regression predicates [23], [29], [31] and adding regression capabilities to the ILP engine [14]. Several systems [3], [17], [18] also produce working prolog programs, that are capable of predicting numerical values, but do this by constructing a regression tree first and then translating it into a Prolog program. By taking this approach some of the power of

hypothesis language is lost.

FORS [14] is the system that first used first order regression. It combined the expressive power of ILP with linear regression. It showed that first order regression is able to produce results. While 4S is based on principles laid by FORS, it is a new system. Several new techniques were developed and some old ones from FORS, that did not perform as expected, were eliminated. In 4S we also included extensive comparing of continuous variables and the option to force the system to count only basically different examples to satisfy the minimal number of examples condition (as described in Section 3.1).

The system described in [29], [30] as well as a system described in [31] are also capable of handling numerical domains using background knowledge for comparisons, and other numerical operations including regression. Since the main goal of the designers was to keep all capabilities of the original ILP engine, several compromises had to be done. In case of numerical domains, those systems cannot navigate efficiently through the search space before predicates predicting numerical values are added to the clause. To ease the navigation through the search space and to produce meaningful clauses, the system requires the user to provide a refinement operator, cost calculation and pruning. This certainly enhances the performance of the system compared to 4S, but it also significantly decreases the ease of use and practical applicability to real-live problems. On the theoretical-ability scale, those systems outrank 4S, but judging from results presented in [29] and [30] one gets the impression that 4S is at least competitive in terms of practical use, as it does not require constant involvement by the user.

6. TEST DOMAINS

6.1. Steel grinding domain

The steel grinding domain data describes the relationship between the roughness of a steel workpiece and the sound produced by grinding that workpiece [13]. The idea is to enable automatic control of the grinding process. An example is termination of the process when performance becomes unsatisfactory [12]. Since control decisions can be quickly deduced from the roughness of the workpiece, our task was to predict roughness from the sound of grinding.

The data was collected during an experiment where the workpiece was fitted with an acceleration sensor. That data was then spectrally analyzed, and the following data was available to 4S:

- size of the area of whole spectrum
- frequency of the maximum peak
- frequency of the middle of the spectrum

During the experiment the process of grinding was stopped and the surface roughness measured 123 times. Background knowledge predicates enabling 4S to compare variables were given (<= and >=).

Table 2 Relative error of 4S on Steel grinding.

min no. of examples	all var. test	two var. test	one var. test	"no" var. test
1	0.633	0.584	0.529	0.840
2	0.656	0.594	0.518	0.629
4	5.403	1.877	1.257	0.595
6	0.550	0.470	0.536	**0.624**
8	0.509	0.461	0.558	0.634
10	0.508	0.452	0.545	0.657
12	0.509	0.476	0.544	0.643
14	0.508	0.473	0.547	0.656
16	0.500	0.465	0.544	0.660
18	0.500	0.453	0.558	0.662
20	**0.441**	**0.451**	0.565	0.662
30	0.443	0.455	0.559	0.655
50	0.501	0.516	**0.499**	0.825
80	0.457	0.464	0.674	0.965
100	1	1	1	1

Table 3 Relative error of FORS on Steel grinding.

min no. of examples	lin reg test	no reg test	lin reg, MDL test	no reg, MDL, test
1	0.74	0.79	0.73	0.62
2	9.15	0.69	0.76	0.60
4	1.01	**0.55**	0.71	0.58
6	0.81	0.60	0.66	**0.54**
8	0.78	0.58	0.63	0.59
10	0.72	0.56	**0.62**	0.56
12	0.69	0.63	0.63	0.64
14	**0.64**	0.60	0.62	0.62
16	0.70	0.57	0.70	0.57
18	0.71	0.62	0.71	0.62
20	0.74	0.64	0.74	0.64

Experiments were conducted in the form of 10 different sets consisting of 70% learning and 30% testing examples (same sets were used by FORS) and all combinations of minimal number of (original) examples in a clause (possible values 1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 30, 50, 80, 100) and the type of linear equation (allowed are all variables, 2, 1 and no variables - a prediction of a constant). From the results on the learning and test data relative (compared to the prediction of an average learning value) average error was calculated. The results are presented in Table 2.

If we compare the results of experiments made with 4S in Table 2 and FORS [15] in Table 3 we can see that when regression is not permitted FORS (with or without MDL) performs better than 4S. However with linear regression even when using MDL FORS overfits the data. The clause with the best results by 4S (made using linear

regression with at least 20 examples covered by each clause) is small and simple, just what we would hope for.

6.2. Finite element mesh domain

Finite element method is used in engineering where stress in structures needs to be calculated. The structure is divided into many simpler elements. A set of linear equations must be solved to calculate the deformations of the elements. Since the number of the elements affects the computational time and the accuracy of the solution (both increase with the number of the elements) a suitable compromise in the partition into elements must be found. In recent years a number of very successful mesh generation algorithms was reported [20]. We should not be surprised to see a significant gap between 4S and specialized algorithms. However, our aim is to compare 4S' ability to generate meshes to that of other ILP systems.

Partitioning usually cuts the edges of the structure into multiple parts. Then the mesh dividing the elements is automatically generated, from the start points on the edges. So the task is to find the number of partitions for each edge of the structure. Since the edges can have different attributes and their partitioning is also depended on other edges and their attributes, inductive logic programming is the natural choice for solving this problem. Others have tried it before and it turned out to work well [5], [7], [8], [14], [16].

In our experiments the data from [8] was used. It describes five structures labeled from A to E. Each structure has from 28 to 96 edges. For each edge a number of partitions was determined and computationally verified by an expert [8]. Our target was a relation that describes the number of elements for each edge.

Experiments consisted of all combinations of parameter values for `min_examples` (2, 5, 10) - only original examples were counted, and `max_literals` (2, 4, 6).

At all times the maximal variable depth was set to 2, which forced 4S to use only the edges at most two steps away from the target edge (neighbors, opposites, neighbor's neighbors…). For each possible setting five experiments were made. Every time one of the structures was set aside for testing and the other four were used for learning. Results of the experiments can be found in Table 4, where a prediction is deemed correct if 4S prediction was correct after rounding to the nearest integer. If the prediction was exactly between two integers (e.g. 1.5) false prediction was assumed.

Figure 2 shows the hypothesis constructed with at least 10 examples and no more than 4 literals in a clause from structures A, B, C, D and tested on structure E. The induction took less than 15 seconds on 700 MHz Pentium III computer running Windows 2000.

We compare results of 4S with results of FOIL [27], mFOIL [10], GOLEM [24], MILP [16], FFOIL [28], FORS [14] and CLAUDIEN [5] in Table 5. Results for FOIL and FFOIL were taken from [28], for GOLEM and MILP from [16], for mFOIL and FORS from [14] and for CLAUDIEN from [5]. We can see that 4S performs better than FOIL and mFOIL, slightly better than GOLEM and CLAUDIEN, similar to MILP, and FORS and worse than FFOIL.

Table 4 Results of FEM domain experiments

Minimal number of examples	2			5			10		
Maximal number of literals	2	4	6	2	4	6	2	4	6
Structure used for testing									
A (55 edges)	22	22	22	22	20	19	21	21	22
B (42 edges)	10	12	12	11	10	7	14	10	13
C (28 edges)	5	10	10	6	5	7	8	8	8
D (57 edges)	16	21	22	14	20	12	14	21	18
E (96 edges)	20	22	22	11	6	3	9	28	4
Total correct (from 278)	73	87	88	64	61	48	66	88	65

Table 5 Comparison to other systems on FEM domain

Structure	FOIL	mFOIL Lapl	mFOIL m=0	GOLEM	MILP	FFOIL	FORS	CLAUDIEN	4S
A	16	23	22	21	21	21	22	31	22
B	9	12	12	12	12	15	12	9	12
C	8	9	9	10	11	11	8	5	10
D	12	6	6	16	16	22	16	19	22
E	16	12	12	21	30	54	29	15	22
Sum	61	62	61	80	90	123	87	79	88
%	22	22	22	29	32	44	31	28	32

7. DISCUSSION

Experiments have shown that the 4S system is capable of combining ILP and numerical regression. There are two major problems: recursion and regression together cause instability in the regression process (when recursion is used, some input arguments in the regression are dependent on output arguments). The other problem is that time complexity, which is already high in ILP, has additionally increased due to additional numeric capabilities. Further research is needed to overcome these problems and move towards a fully applicable system.

The severity of both problems can be reduced. The time complexity calls for better heuristics and optimization of the code, for example in the ILP part where construction of equivalent clauses can be avoided. The problem with recursion and

regression demands more effort. Usually ILP systems require a complete set of examples to successfully induce recursive definitions. However, a complete set of examples can be expected only in artificial domains and not in real world domains. That is why we need to create a process that combines available examples and simulates the missing examples.

In summary, 4S performs as well as other academic systems dealing with reasonable problem domains. If we compare accuracy with FORS, 4S performs similarly or even marginally better on some domains, while being more flexible and tunable to a particular domain. In comparison with more complex systems that demand inclusion of additional program code, setting parameters in 4S is far less demanding even for non-specialized users.

REFERENCES

[1] A. Appice, M. Ceci, and D. Malerba. Mining model trees: A multi-relational approach. In T. Horvath and A. Yamamoto, editors, Inductive Logic Programming, 13th International Conference, ILP 2003, volume 2835 of LNAI, pages 4_21. Springer-Verlag, 2003

[2] Bratko, I. and Džeroski, S., "Engineering applications of ILP", *New Generation Computing 13* (pp. 313-333), 1995.

[3] Blockeel, H., *Top-down induction of first order logical decision trees*. PhD Thesis, Katholike Universiteit, Leuven, Belgium, 1998.

[4] Camacho, R., "Learning stage transition rules with Indlog", *Proceedings of The Fifth International Workshop on Inductive Logic Programming (ILP-94)* (pp,273-290), Bad Honnef/Bonn, Germany: Gesellschaft für Mathematik und Datenverarbeitung Sankt Augustin, 1995.

[5] Dehaspe, L., van Laer, W., and De Raedt, L., "Applications of a logical discovery engine", in *Proceedings of the Fourth International Workshop on Inductive Logic Programming (ILP-94)*, GMD-Studien Nr. 237, 1994.

[6] Demšar, D., *Dealing witn Numerical Problems Using Inductive Logic Programming*, Master's Thesis, University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia, in Slovene, 1999.

[7] Dolšak, B., and Muggleton, S., "The application of inductive logic programming to finite-element mesh design", Stephen Muggleton. (ed.), *Inductive Logic Programming*: Academic Press, 1992.

[8] Dolsak, B., Bratko, I. and Jezemik, "A, 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, 1994.

[9] Džeroski, S., and Todorovski, L., "Discovering dynamics: from inductive logic programming to machine discovery", *Journal of Intelligent Information Systems*, 4.89-108, 1995.

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

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

[12] Filipič, B., Junkar, M., Bratko, I. and Karaliè, A, "An application of machine learning to a metal-working process", in *Proceedings of IT-91*, Cavtat, Croatia, 1991.

[13] Junkar, M., Filipič, B. and Bratko, I., "Identifying the Grinding Process by Means of Inductive Machine Learning", *Computers in Industry*, 17(2-3), 147-153, 1991.

[14] Karalič, A. and Bratko, I., "First Order regression". In *Machine Learning*, Volume 26, Numbers 2/3. Kluwer Academic Publishers, 1997.

[15] Karalič, A., *First Order Regression*. PhD thesis, University of Ljubljana, Faculty for Electrical Engineering and Computer Science, Ljubljana, Slovenia, 1995.

[16] Kovačič, M., *Stochastic Inductive Logic Programming*, PhD thesis, Universty of Ljubljana,, Faculty of Electrical Engineering and Computer Science, Ljubljana, Slovenia, 1995

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

[18] S. Kramer. Relational Learning vs. Propositionalization: Investigations in Inductive Logic Programming and Propositional Machine Learning. PhD thesis Vienna University of Technology, Vienna, Austria, 1999

[19] Križman, V., *Avtomatic discovery of structure in dynamic systems models*, PhD Thesis, University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia, in Slovene, 1998.

[20] Lavrač, N. and Džeroski, S., *Inductive Logic Programming - Techniques and Applications*, 1994.

[21] Miller, G. L.,Teng, S. H., Thurston, W. A., Vavasis, S. A. "Graph Theory and Sparse Matrix Computation", *The IMA Volumes in Mathematics and its Applications 56*, (pp 57-84), Springer-Verlag, 1993.

[22] Mizoguchi, F. and Ohwada, H., "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), Katholike Universiteit Leuven, Heverlee, Belgium, 1995.

[23] Muggleton S., Page D., "Beyond first-order learning: inductive learning with higher order logic", *PRGTR 13-94*, Oxford University Computing Laboratory, Oxford, 1994.

[24] Muggleton., S. and Feng, C. "Efficient induction or logic programs", *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, Japan, 1990.

[25] Page, C. D., Frisch, A. M., "Generalization and learnability: A study of constrained atoms", in: S. Muggleton (Ed.), *Inductive Logic Programming* (pp 29-56), Academic Press, London, 1992.

[26] Quinlan, R. and Cameron-Jones, R. M., "Efficient top-down induction of logic programs", *SIGART Bulletin 5* (1) 33-42, 1994.

[27] Quinlan, R., "Learning logical definitions from relations", in *Machine Learning* 3(5), 1990.

[28] Quinlan, R., "Learning First-Order Definitions of Functions", *Journal of Artificial Intelligence Research*, volume 5, (pp 139-161), 1996.

[29] Srinivasan, A. and Camacho, R., "Numerical reasoning with ILP system capable of lazy evaluation and customized search", *The Journal of Logic Programming*, Volume 40, Numbers 2/3, 1999.

[30] Srinivasan, A., "Experiments in numerical reasoning with ILP", Michie, D., Muggleton, S., and Furukawa, K., (ed.), *Machine Intelligence* 15, Oxford University Press, 1996.

[31] Srinivasan, A. The Aleph Manual. Technical Report, Computing Laboratory, Oxford University, 2000,,