Learning in Constraint Databases

Teddy Turmeaux, Christel Vrain

LIFO - Université d'Orléans - BP 6759
45067 Orléans Cedex 2 - France
email : {turmeaux,cv}@lifo.univ-orleans.fr

Abstract. For several years, Inductive Logic Programming (ILP) has been developed into two main directions: on one hand, the classical symbolic framework of ILP has been extended to deal with numeric values and a few works have emerged, stating that an interesting domain for modeling symbolic and numeric values in ILP was Constraint Logic Programming; on the other hand, applications of ILP in the context of Data Mining have been developed, with the benefit that ILP systems were able to deal with databases composed of several relations.

In this paper, we propose a new framework for learning, expressed in terms of Constraint Databases: from the point of view of ILP, it gives a uniform way to deal with symbolic/numeric values and it extends the classical framework by allowing the representation of infinite sets of positive/negative examples; from the point of view of Data Mining, it can be applied not only to relational databases, but also to spatial databases.
A prototype has been implemented and experiments are currently in progress.

1 Introduction

In Inductive Logic Programming, most discrimination learning tasks can be stated as follows: given a domain theory defining basic predicates $q_1, \ldots, q_l$, a set of target predicates $p_1, \ldots, p_m$ defined by positive and negative examples, find a logic program, expressed with the predicates $q_1, \ldots, q_l, p_1, \ldots, p_m$, defining the target predicates, covering their positive examples and rejecting their negative ones.

Examples are usually expressed by ground atoms $p_i(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n$ denote constants. The domain theory is either extensionally defined by a set of ground atoms, or intentionally defined by a logic program. In this last case, a model, expressed by a set of ground atoms under closed world assumption, is computed, and only this model is used for learning. Nevertheless, when the representation language contains function symbols other than constants, this model may be infinite. To overcome this problem, either a finite, but incomplete, model is computed, usually by limiting the depths of the derivation trees, or the underlying representation language is merely restricted to *Datalog* (or $Datalog^{Neg}$) rules, meaning that no function symbols but constants are used. Nevertheless, this leads to important drawbacks:

- Function symbols must be translated into predicate symbols expressing the graph of the function. For instance, the function *successor* must be represented by a binary predicate $succ(X, Y)$ ($succ(X, Y)$ is true, when Y is the successor of X),
- A finite set of constants is given, thus when dealing with numeric values, compelling to provide a bound on these values. If we consider again, the successor function and if we suppose that the domain is restricted to $\{0, 1, 2, 3\}$, we must either express that 3 has no successor, or introduce, as done in the system Foil [11], a new value ω expressing that the successor of 3 is out of range.

Such requirements can lead to learn logic programs which take advantage of the closed world nature of the representation and which are incorrect when applied outside the finite domain. To deal with this problem, mostly generated by numeric values, it has been proposed [10, 12, 13, 1, 9, 7, 3] to model the learning task in the field of Constraint Logic Programming (**CLP**) [4]. We propose a new framework which is also based on the notion of constraints, but which takes into account the database dimension, by means of Constraint Databases [5].

This new domain has emerged in the field of Databases to benefit from the declarative power of constraints and from the introduction of domain theories to express relations between variables. The use of constraints is very important for modeling concepts the extensions of which are infinite, and therefore Constraint Databases are of particular interest for modeling spatial or temporal information. In this paper, we present a new model for Inductive Logic Programming that enables to deal with numeric and symbolic information: examples are represented by conjunctions of constraints and the domain theory is also expressed by sets of constraints. This model generalizes the classical one: a target concept defined by a predicate p with n places can be viewed as a relation p over n variables, namely $X_1, \ldots, X_n$, an example usually expressed by a ground atom $p(a_1, \ldots, a_n)$ can be viewed as a generalized tuple $X_1 = a_1 \wedge \ldots \wedge X_n = a_n$ and, when a model of the domain theory is computed before learning, this model expressed by a set of ground atoms can also be viewed as a set of constraints.

The paper is organized as follows. Section 2 is devoted to the basic notions of Constraint Databases. In Section 3, we propose a framework for learning in Constraint Databases and the underlying notions of coverage and generality are defined. The system we have developed is then presented (Section 4).

2 Constraint Databases

2.1 Framework

From the point of view of CLP, Constraint Databases represent another way of considering constraints, which take into account the notion of storing data. As in CLP, it is built on a language for expressing constraints. Let us denote by Π_c (respectively Σ_c) the set of constrained predicate (respectively function) symbols

used for expressing constraints and $\Sigma = (\Sigma_c, \Pi_c)$. We assume that Π_c contains at least the equality operator (=) between variables, or between a variable and a constant of the domain. Terms and constraints are syntactically defined as follows:

A *term* is a variable or a constant; if f is a function symbol of Σ_c, with arity n and if $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.
A *constraint* is an expression $p(t_1, \ldots, t_n)$, where $p \in \Pi_c$ and $t_1, \ldots, t_n$ are terms.

To compute the semantic of constraints, a Σ-structure $\mathcal{D}$ must be defined. It is composed of a set D representing the domain of the constraints, and for each function symbol of Σ_c and for each predicate symbol of Π_c, an assignment of functions and relations on the domain D which respect the arities of the symbols. $\mathcal{D}$ is referred as to the *domain of computation.*

From the point of view of Databases, Constraint Databases extend both relational, deductive and spatial databases, allowing the definition of infinite sets by the introduction of generalized k-tuples, as in [6].

A *generalized k-tuple* t is a quantifier-free conjunction of constraints, over a set $X_1, \ldots, X_k$ of variables (the variables occurring in the generalized tuple t). For sake of simplicity, in the following, $\tilde{X}$ will denote the variables $X_1, \ldots, X_k$.

For instance, let us consider $\Pi_c = \{=, \leq, <\}$ and $\Sigma_c = \{0, 1, +\}$. Let us write n for the abbreviation of $1 + \ldots + 1$, n times. Let D be the set of integers and $\mathcal{D}$ the Σ-structure over D, which interprets the symbol of Σ as usual.
$X \leq 3 \wedge 5 \leq Y \wedge Y < 8$ is a generalized 2-tuple over X and Y.

A *generalized relation* R of arity k over the variables $X_1, \ldots, X_k$ is a finite set of generalized k-tuples over $X_1, \ldots, X_k$. Therefore, it is a disjunction of conjunctions of constraints.

For instance, a generalized relation R over the variables X and Y can be defined by $\{ (X \leq 3 \wedge 5 \leq Y \wedge Y \leq 8), (10 \leq X \wedge Y \leq 20) \}$. Hence a generalized tuple of arity k is a finite representation of a possibly infinite set of tuples of arity k.

Let us notice that, if we consider a classical relation R defined on the variables X and Y, then for instance, the tuple $R(3, 4)$ can be represented by the generalized 2-tuple $X = 3 \wedge Y = 4$, and the whole relation R can be viewed as a finite set of such generalized 2-tuples. .

Finally, a *constraint database* (or *generalized database*) is a finite set of generalized relations.

Let $\mathcal{D}$ be the domain of computation. The *extension of a generalized k-tuple* t over $X_1, \ldots, X_k$, denoted by $ext(t)$ is the set of tuples $< a_1, \ldots, a_k >$ such that the valuation $X_1 = a_1, \ldots, X_k = a_k$ satisfies t.

In the same way, the extension of a relation R is defined by:

$$ext(R) = \cup_{t \in R} ext(t)$$

Let us notice that the extension of a generalized tuple t or, of a generalized relation R, may be infinite.

Codd's relational algebra had been extended to constraint databases. The classical algebraic operations, such as join, projection, selection, ... are defined in terms of constraints, for a specific domain of computation [2].

2.2 Join

Let R_1 be a generalized relation over variables $\tilde{X}$, and R_2 be a generalized relation over variables $\tilde{Y}$, and $\tilde{Z} = \tilde{X} \cup \tilde{Y}$, then the *natural join* of R_1 and R_2, denoted by $R_1 \bowtie R_2$, is defined on $\tilde{Z}$ by:

$$R_1 \bowtie R_2 = \{t_1 \wedge t_2 | t_1 \in R_1, t_2 \in R_2, ext(t_1 \wedge t_2) \neq \emptyset\}$$

Example 1. Let *Circles* be a relation over the variables ID_{Circle}, X and Y, defined as follows $\{(ID_{Circle} = 1 \wedge (X-2)^2 + Y^2 < 1), (ID_{Circle} = 2 \wedge (X+3)^2 + Y^2 < 4)\}$. In such a relation a tuple, as for instance $ID_{Circle} = 1 \wedge (X-2)^2 + Y^2 < 1$ represents the set of points (X, Y) belonging to the circle with index 1 and defined by the equation $(X-2)^2 + Y^2 < 1$. Let us consider also the relation *Rectangles* defined by $\{ID_{Rect} = 1 \wedge 0 < X < 2 \wedge 0 < Y < 25\}$ then $Intersect = Circles \bowtie Rectangles = \{ID_{Circle} = 1 \wedge (X-2)^2 + Y^2 < 1 \wedge ID_{Rect} = 1 \wedge 0 < X < 2 \wedge 0 < Y < 25\}$ represents a relation over ID_{Circle}, ID_{Rect}, X and Y, such that the circle and the rectangle respectively defined by ID_{Circle} and ID_{Rect} intersect and the points defined by (X, Y) belong to that intersection. Let us insist on the fact that the tuple $ID_{Circle} = 2 \wedge (X+3)^2 + Y^2 < 4 \wedge ID_{Rect} = 1 \wedge 0 < X < 2 \wedge 0 < Y < 25$ does not belong to $Intersect$, since its extension is empty.

2.3 Selection

The selection of a relation R (over variables $\tilde{X}$) on a constraint c over variables $\tilde{Y}$, with $\tilde{Y} \subseteq \tilde{X}$, denoted by $\sigma_c(R)$, is defined by:

$$\sigma_c(R) = \{t \wedge c | t \in R, ext(t \wedge c) \neq \emptyset\}$$

Example 2. Let us consider again the relation defined in Example 1.

$\sigma_{(ID_{Circle}=2)}(Intersect) = \{\}$

$\sigma_{(X<-2)}(Circle) = \{ID_{Circle} = 2 \wedge (X+3)^2 + Y^2 < 4 \wedge X < -2\}$.

The relation $\sigma_{(X<-2)}(Circle)$ contains a unique tuple. It represents the area of the second circle, for which $X < -2$. The first circle has been deleted, since no point in it satisfies $X < -2$.

2.4 Projection

If R is a generalized relation over variables $\tilde{X}$, and $\tilde{Z} \subseteq \tilde{X}$, then the projection of R on $\tilde{Z}$ denoted by $\Pi_{\tilde{Z}}(R)$ is defined by:

$$\Pi_{\tilde{Z}}(R) = \{\Pi_{\tilde{Z}}(t) | t \in R\}$$

where $\Pi_{\tilde{Z}}(t)$ represents the projection of the tuple t onto the variables $\tilde{Z}$.

Example 3. With *Circles* defined in Example 1, $\Pi_{[ID_{circle},X]}(Circles)$ is $\{(ID_{circle} = 1 \wedge 1 < X < 3), (ID_{circle} = 2 \wedge -5 < X < -1)\}$.

Example 4. Let us consider now the relation R defined by $\{(X = Z + 1 \wedge Y = Z + 2), (X = 2)\}$ then $\Pi_{[X,Y]}(R) = \{(X = Y - 1), (X = 2)\}$.

2.5 Union

The union of two relations is defined by :

$$R_1 \cup R_2 = \{t | t \in R_1 \text{ or } t \in R_2\}$$

3 A new framework for Learning in Constraint Databases

3.1 Coverage - Generality order

A generalized k-tuple t over the variables $X_1, \ldots, X_k$ is *satisfiable* if there exists an assignment α of elements of D to the variables of t such that $\alpha.t$ is evaluated to *true*. In other words, it is satisfiable, when its extension is not empty.

We can now compare a generalized tuple over variables $\tilde{X}$ and a generalized relation over variables $\tilde{Y}$.

A generalized tuple t over variables $\tilde{X}$ is *covered* by a generalized relation R over variables $\tilde{Y}$, when $ext(\Pi_{\tilde{X} \cap \tilde{Y}}(t)) \subseteq ext(\Pi_{\tilde{X} \cap \tilde{Y}}(R))$.

For instance, let us consider the generalized relation R, composed of the unique tuple $X \leq 1$. The tuple $X = 1 \wedge Y = 1$ is *covered* by the relation R.

A generalized relation R is covered by a generalized relation R', if $\forall t \in R$, t is covered by R'. In that case, R' is said to be more *general* than R (R is more *specific* than R').

Let us notice that this definition is not equivalent to the following one: for any tuple t of R, there exists a tuple t' of R', such that t is covered by t'.

Satisfiability, Consistency checking When a generalized tuple t is grounded, i.e., when $|ext(t)| = 1$, then t is covered by a relation R if there exists t_R in R, such that the constraint $t \wedge t_R$ is satisfiable. Therefore, in that case, testing whether a tuple is covered or uncovered requires a complete satisfiability test (a satisfiability test is complete if for every constraint, the test answers *yes* if the constraint is satisfiable, or *false* otherwise).

For some domains, as for instance, linear real equality constraints, finite tree-constraints, or sequence constraints, complete solvers exist. Nevertheless, it may happen that a complete solver exists, but is untractable, and that for reasons of efficiency, an incomplete solver is used.

3.2 Learning setting

Given:

- a generalized relation R_T over a set of variables, composed of generalized tuples that can be labelled either as positive tuples, or as negative tuples. Let us insist on the fact that a positive (respectively negative) tuple represents a possibly infinite set of positive (respectively negative) examples.
- a concept description language $\mathcal{L}$, specifying syntactic restrictions on the intensional definition of the target relation R_T,
- a set BK of generalized relations R_i, $i = 1, \ldots, n$, specifying background knowledge and which can be used in the definition of R_T,

Find:

- a generalized relations R_H expressed in terms of (generalized-)relational operations that is complete and consistent, when completeness and consistency are defined as follows.
  Completeness: R_H must cover every positive tuples in R_T
  Consistency: R_H must cover no negative tuples in R_T.

This framework generalizes the classical ILP setting, since the representation of tuples by means of constraints enables to represent infinite models. For instance, the relation *Circles* defined in Example 1 represents for each circle the set of points belonging to it, and such a set is infinite. In the same way, infinite sets of positive and negative examples can be expressed and such examples have not to be ground. Let us also notice that some variables of the relation may not appear in every tuple. For instance, the two tuples $< true, true >, < true, false >$ of a binary relation can be viewed as the generalized tuple $X = true$.

4 Our system

4.1 The Learning Algorithm

Our system uses a divide-and-conquer method, as the one used in the system Foil [11]: it learns a generalized relation covering some positive tuples, it removes

the covered positive tuples, and the process is repeated until no more positive examples remains. Let POS and NEG respectively represent the positive and negative generalized tuples and let $\rho(R_C)$ represents a refinement operator, which takes as input a relation R_C and which gives a set of relations more specific than R_C.

Algorithm.

1. $R_H = \emptyset$
2. $POS =$ positive tuples of R_T
3. $NEG =$ negative tuples of R_T
4. While $POS \neq \emptyset$
   (a) $R_C = \emptyset$
   (b) While $cov(R_C, NEG) \neq \emptyset$
      i. $R_C =$ best relation in $\rho(R_C)$
   (c) $POS = POS - cov(R_C, POS)$
   (d) $R_H = R_H \cup R_C$

The main steps in that algorithm are the computation of $\rho(R_H)$ and the choice of the best relation.

Definition (Refinement operator). Let $\mathcal{R}$ be a set of generalized relations, that are supposed to be over set of variables that are disjoint ($\forall R,\ S \in \mathcal{R}, R \neq S \rightarrow var(R) \cap var(S) = \emptyset$). Let $R \in \mathcal{R}$.

$$\rho(R) = \{R' | \exists S \in \mathcal{R}, \exists Z \subseteq var(S), \exists \alpha Z \rightarrow var(R), R' = R \bowtie \alpha.S\}$$

In the following, $\mathcal{R}$ is composed of

- background generalized relations provided by the user before learning,
- or domain specific relations that are induced, for instance, for expressing a constraint between variables.

In the current implementation, symbolic and numeric domains are ordered and we search for constraints $X \leq \theta$ for each variable X occurring in the relation. Moreover, we search for linear inequalities between numeric variables, as done in the system ICC [8].

Let us insist on the fact that Constraint Databases truly provide a uniform way to deal with these different relations. For example, if we need to introduce a constraint, as for instance $X \leq 2$, to discriminate between positive and negative tuples, such a constraint can be viewed as a generalized relation over the variable X defined by only one generalized tuple, namely $X \leq 2$. A linear constraint between variables $X_1, \ldots, X_n$, can be represented by a generalized relation over variables $X_1, \ldots, X_n$ with a unique generalized tuple, namely, that linear constraint.

Property. Let R be a generalized relation. $\forall R' \in \rho(R)$, $R' \leq R$ (R' is more specific than R).

Example 5. Let us illustrate this process by a very simple example. Let us consider the target relation R_T, defined on the variables X, Y and Z by:

(+)	$X = 0 \wedge Y = 0 \wedge Z = 0$	(−)	$X = -1 \wedge Y = 0 \wedge Z = 0$
(+)	$X = 1 \wedge Y = 0 \wedge Z = 0$	(−)	$X = 3 \wedge Y = 0 \wedge Z = 0$
(+)	$X = 2 \wedge Y = 0 \wedge Z = 1$		

Let us suppose also that we have the background relation $dec(A, B)$, defined by the generalized tuple: $B = A - 1$.

At the beginning of the learning process, the hypothesis relation R_H is empty. In our simple example, there is only one possible relation to join to R_H, namely $dec(A, B)$. There are many possible bindings, as for instance $A = X \wedge B = Y$, or $A = X \wedge B = Z$. Let us suppose that we choose the second binding. Then, the hypothesis R_H is set to $dec(X, Z)$. In that case, since the target relation is grounded (each generalized tuple has a unique solution), an example e of R_T is covered by R_H, when $e \wedge Z = X - 1$ is satisfiable. Only the two last positive examples of R_T are covered and there remains a negative example (the first one). We get the following set of positive and negative examples.

(+)	$X = 1 \wedge Y = 0 \wedge Z = 0$	(−)	$X = -1 \wedge Y = 0 \wedge Z = 0$
(+)	$X = 2 \wedge Y = 0 \wedge Z = 1$		

After variable bindings, the system searches for a constraint that would enable to discriminate positive from negative examples. A simple constraint would be $X \geq 0$. The target relation R_H is set to $dec(X, Z) \wedge X \geq 0$, it is defined by the only generalized tuple $Z = X - 1 \wedge X \geq 0$. The two positive examples are covered (for instance, the first one is covered since $X = 1 \wedge Y = 0 \wedge Z = 0 \wedge Z = X - 1 \wedge X \geq 0$) is satisfiable); the negative example is no longer covered.

The hypothesis that has been learned for our concept is therefore: $dec(X, Z) \wedge X \geq 0$. It does not cover the first positive example and therefore if we want the learned definition to be complete, a new hypothesis must be learned for covering it.

Best candidate relation. For the time being, we have chosen a strategy, close to that developed in the system Foil: a divide-and-conquer strategy to cover all the positive examples and a refinement process to build a good hypothesis covering as many positive examples as possible and rejecting all the negative examples. This refinement process is based on different choices: a relation, a binding and a constraint. To achieve such choices, different measures have been introduced in ILP, taking into account the number of positive and negative examples covered. In our framework, the number of positive and negative generalized tuples that are covered can be counted. Nevertheless, a generalized tuple can represent an infinite set of examples and counting the number of generalized tuples may be a bad approximation. For the time being, to overcome this problem, we have implemented a semi-automatic choice procedure:

1. the interest of each relation is computed with the formula

$$(n^+ + n^-)/n$$

where n^+ (resp. n^-) denotes the number of positive (respectively negative) generalized tuples of the target relation that are covered (respectively uncovered) by the current hypothesis, and n is the total number of generalized tuples in the target relation,

2. the user chooses among the relations that have a high interest.

4.2 Efficiency

Biases. In order to reduce the number of possible variable bindings, two classical ILP biases have been implemented:

- typing and,
- usage declaration: the system reduces the hypothesis space by using user supplied information about the mode of the variables, which is given when specifying the background relations:
  (+) indicates that this position must contain a variable already present in the relation,
  (−) indicates that this position must contain a variable that is not already present.
  For instance, the declaration $add(+X : integer, +Y : integer, -Z : integer)$ specifies for the generalized relation *add* over the variables X, Y and Z that, when introduced in a hypothesis, X and Y must already occur in that hypothesis.

Speed-up. For some domains and some kinds of constraints, the efficiency of the learning system can be improved by normalizing tuples, that is to say by rewriting tuples into a unique, canonical form. Such a process allows both to speed-up the constraints satisfiability test and to remove redundant constraints. For instance, [6] have proposed a canonical representation of tuples for dense order constraints (dense order inequality constraints are formulas of the form $x\theta y$ and $x\theta c$ where x, y are variables, c is a constant, and θ is one of $=, \leq, <$). This representation is based on the storage of a higher bound and of a lower bound for each variable of the generalized tuple.

In our system, numeric and symbolic domains are ordered and to each constraint is associated a higher and a lower bound. This allows a very fast induction process for finding interesting thresholds between positive and negative tuples.

5 Preliminary results

We are currently testing our system on a dataset, which had been generated to test the system GKS [10], an ILP system developed for the induction of constraint logic programs.

The dataset provides symbolic and numerical information about spatial layout of Japanese houses, e.g. the size of the rooms, their spatial relations or the total size of the house. The background knowledge is composed of both symbolic and numeric information.

We have transformed the dataset into a constraint database. For instance, the relation *east-of* that indicates the relative position of two rooms in a house is described as follows (a disjunction of conjunctions):

$east-of(\ +house:\ integer, room1:\ room, room2:\ room\ )$
[house=1∧room1=living-dining∧room2=kitchen]
[house=1∧room1=toilet∧room2=bath]
[house=1∧room1=entrance∧room2=hall]
...
The complete database contains about 85000 tuples.

Let us consider, for instance, the predicate *corner(H, living, south-east)* meaning that in the house represented by H, the living room is located in the direction south-east. Such a concept has 32 (tuple-)examples which are all grounded. At the beginning, the system builds some hypotheses like:

Hypothesis	Number of positive examples covered	Number of negative examples rejected
south-of$(H, R_{11}, R_{12}) \bowtie \{R_{12} = living\}$	1	all
south-of$(H, R_{11}, R_{12}) \bowtie \{R_{11} = living\}$	16	4
...	...	...

Since background knowledge contains relations involving real variables, tuples are stored in a canonical form, expressing the minimum and the maximum values of each real variable. At any time, the system performs a search for interesting thresholds, discriminating positive from negative examples. This enables to learn the following rules which are identical to those induced by the GKS system (*ent-direction* represents the direction of the entrance):

Induced rule	$\oplus$ covered	$\ominus$ rejected
area$(H, A) \bowtie \{A \leq 58.58\}$	3	all
ent-direction$(H, D) \bowtie$ area$(H, A) \bowtie \{A \leq 60.43 \wedge D = west\}$	3	all

We have started a systematic comparison of the induced rules.

6 Conclusion

We have proposed a new framework for learning, expressed in terms of Constraint Databases, that generalizes the classical ILP framework in several directions:

- it allows the representation of infinite sets of examples by means of constraints,
- it is expressed in terms of Databases and therefore it can been straightforwardly applied to Data Mining tasks,
- it can be applied to spatial databases.

Nevertheless, the system that we have developed must be improved:

The form of the gain. As already explained, our measure has one main drawback: it does not take into account the fact that a generalized tuple can cover an infinite set of examples. To deal with this problem, new measures must be developed, for instance based on the size of the example space, represented by a generalized tuple.

Biases. New biases must be implemented to reduce the search space.

Induction of constraints. For the time being, the constraints that are induced are mostly thresolds for numeric and symbolic variables (let us recall that an order is given even on symbolic domains), and linear inequalities. The system must be extended to induce other kinds of constraints.

Spatial Databases. The system is currently tested on classical ILP problems. It must be tested on learning in Spatial Databases, that cannot be directly handled by ILP systems.

References

1. Simon Anthony and Alan M. Frisch. Generating numerical literals during refinement. In Nada Lavrač and Sašo Džeroski, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *LNAI*, pages 61–76, Berlin, September 17–20 1997. Springer.
2. B. Catania, A. Belussi, and E. Berttrio. Introducing external functions in constraint query languages. *Lecture Notes in Computer Science*, 1520:132–??, 1998.
3. Sašo Džeroski, Ljupčo Todorovski, and Tanja Urbančič. Handling real numbers in inductive logic programming: A step towards better behavioural clones. In Nada Lavrač and Stefan Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *LNAI*, pages 283–286, Berlin, April 1995. Springer.
4. Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *The Journal of Logic Programming*, 19 & 20:503–582, May 1994.
5. P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. In ACM, editor, *PODS '90. Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems: April 2–4, 1990, Nashville, Tennessee*, volume 51(1) of *Journal of Computer and Systems Sciences*, pages 299–313, New York, NY 10036, USA, 1990. ACM Press.
6. Paris C. Kanellakis and Dina Q. Goldin. Constraint programming and database query languages. *Lecture Notes in Computer Science*, 789:96–??, 1994.

7. T. Kawamura and K. Furukawa. Towards inductive generalization in constraint logic programs. Technical report, Faculy of Environmental Information, Keio University, 1993.
8. L. Martin and C. Vrain. Efficient induction of numerical constraints. In Zbigniew W. Raś and Andrzej Skowron, editors, *Proceedings of the 10th International Symposium on Foundations of Intelligent Systems (ISMIS-97)*, volume 1325 of *LNAI*, pages 167–176, Berlin, October 15–18 1997. Springer.
9. Lionel Martin and Christel Vrain. Learning linear constraints in inductive logic programming. In Maarten van Someren and Gerhard Widmer, editors, *Proceedings of the 9th European Conference on Machine Learning*, volume 1224 of *LNAI*, pages 162–169, Berlin, April 24–24 1997. Springer.
10. F. Mizoguchi and H. Ohwada. An inductive logic programming approach to constraint acquisition for constraint-based problem solving. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 297–322. Department of Computer Science, Katholieke Universiteit Leuven, 1995.
11. J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In Pavel B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667 of *LNAI*, pages 3–20, Vienna, Austria, April 1993. Springer Verlag.
12. C. Rouveirol and M. Sebag. Constraint inductive logic programming. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 181–198. Department of Computer Science, Katholieke Universiteit Leuven, 1995.
13. Michèle Sebag, Céline Rouveirol, and Jean-François Puget. Induction of constraint logic programs. In Grigoris Antoniou, Aditya K. Ghose, and Mirosław Truszczyński, editors, *Proceedings of the PRICAI '96 Workshops on Reasoning with Incomplete and Changing Information and on Inducing Complex Representations : Learning and Reasoning with Complex Representations (Pricai$_W$ − 96)*, volume 1359 of *LNAI*, pages 148–167, Berlin, August 26–30 1998. Springer.