Polynomial-time Learning in Logic Programming and Constraint Logic Programming

Michèle Sebag[1] and Céline Rouveirol[2]

(1) LMS – URA CNRS 317, Ecole Polytechnique, F-91128 Palaiseau Cedex, Michele.Sebag@polytechnique.fr

(2) LRI – URA CNRS 410, Université Paris Sud, F-91405 Orsay Cedex, Celine.Rouveirol@lri.fr

Abstract. Induction in first-order logic languages suffers from an additional factor of complexity compared to induction in attribute-value languages: the number of possible matchings between a candidate hypothesis and a training example.
This paper investigates the use of a stochastic bias to control this factor of complexity: the exhaustive exploration of the matching space is replaced by considering a fixed number of matchings, obtained by random sampling. One thereby constructs from positive and negative examples a theory which is only approximately consistent. Both the degree of approximation and the computational cost of induction are controlled from the number of samples allowed.
This approach is illustrated and validated on the mutagenesis problem. An ad hoc sampling mechanism has been purposely designed, and experimental results fully demonstrates the power of stochastic approximate induction, in terms of both predictive accuracy and computational cost. Furthermore, this approach applies for learning both logic programs (as it is usually the case in ILP) and constrained logic programs, i.e. extended logic programs that can naturally handle numerical information. The gain of learning constrained logic programs for the mutagenesis problem is evaluated by comparing the predictive accuracy of the theories induced in both languages.

1 Introduction

The framework of Inductive Logic Programming (ILP) [21] allows induction to handle relational problems. This very expressive formalism however raises two major questions: that of dealing with numerical values, and that of mastering the computational complexity pertaining to first-order logic.

Handling numbers in ILP has mainly been tackled via transformation of relational problems into propositional ones *a la LINUS* [15] (see also [37]), or by using adequate "numerical knowledge", be it built-in as in *FOIL* [25] or provided in declarative form as in *PROGOL* [20]. A third possibility is based on Constraint Logic Programming (CLP), which both subsumes logic programming (LP) and allows for the interpretation of predefined predicates, in particular predicates involving numerical variables [8]. An earlier work [30] has presented a learner named *ICP* for *Inductive Constraint Programming*, which uses constraints to prevent negative examples from matching candidate hypotheses.

This paper is concerned with bridging the gap between the above theoretical approach and real-world problems. A major difficulty is that of computational complexity. This general difficulty of induction is usually handled through language biases (e.g. GOLEM considers ij-determinate clauses [22]; *PROGOL* sets an upper bound on the number of literals in a candidate clause [20]) or search biases, (e.g. *FOIL* considers one literal at a time [25]; *FOCL* restricts the amount of look-ahead [24]; *FOIL* and *PROGOL* respectively use the quantity of information and the MDL principle to sort the candidate hypotheses). However, adjusting these biases requires a precise *a priori* knowledge, which is often far from available.
This is the reason why our previous works [26, 29, 30] were based on a variant of the "bias-free" Version Space framework [18], called *Disjunctive Version Space* (DiVS) [28]. However, this gets intractable on truly relational problems, for the number of possible matchings between a hypothesis and a negative example is exponential in the size of the examples; e.g. in the mutagenicity problem, where molecules involve up to 40 atoms, the number of possible matchings goes to 40^{40}.

We therefore propose a new algorithm that builds approximate version spaces with polynomial complexity. This algorithm, named *STILL* for *Stochastic Inductive Learner*, combines Disjunctive Version Spaces with a sampling mechanism: it only considers *some samples of the possible matchings* between a hypothesis and a negative example. It thereby constructs a theory which is only approximately consistent with linear complexity in the number of samples allowed. Besides, classification heuristics taken from the propositional *DiVS* [28] can directly be adapted for *STILL* to cope with noisy and sparse data, while keeping a polynomial classification.

The sampling mechanism allows the expert to control both the computational cost of induction and the degree of approximation of the induced theory, via the number of samples allowed. This heuristics can be used whenever several (many) matchings between a hypothesis and a training example are possible. We study its effects on learning either definite or constrained programs.

The paper is organized as follows. Next section briefly reviews the *Disjunctive Version Space* approach. As the computational pitfall in first-order logic becomes obvious, section 3 discusses how to restrict the matching search space, and introduces the sampling mechanism implemented in *STILL*. The algorithms of induction and classification in *STILL* are given, together with the corresponding polynomial complexity results. Section 4 is devoted to experimental validation on the mutagenicity problem; *STILL* results compare favorably to those of *PROGOL* and *FOIL*, reported from [33]. Finally, some avenues for further research are discussed in section 5.

2 Disjunctive Version Spaces in LP and CLP

This section illustrates the Disjunctive Version Space approach on a problem pertaining to organic chemistry [11]: the mutagenicity problem is one most famous testbed in ILP [33, 10]. A more detailed presentation of Disjunctive Version Spaces in the frame of attribute-value and CLP languages can be found in [28] and [30].

2.1 Data and language of examples

The mutagenicity problem consists in discriminating organic molecules (nitroaromatic compounds) depending on their mutagenic activity (*active* or *inactive*). This still open problem is of utmost practical interest, for these compounds occur in car exhaust fumes, and high mutagenic activity is considered carcinogenic.

The description of molecules considered in this paper includes the description of atoms and bonds, augmented with non structural information (five boolean and numerical attributes) measuring the hydrophobicity of the molecule, the energy of the molecule lowest unoccupied molecular orbital, and so on. A molecule a is thus described by a ground clause, an excerpt of which is:

$$\begin{aligned} tc(a) :- \ & atom(a, a_1, carbon, 22, -0.138), \cdots, atom(a, a_{26}, oxygen, 40, -0.388), \cdots \\ & bond(a, a_1, a_2, 7), \cdots, bond(a, a_{24}, a_{26}, 2), \\ & logp(a, 4.23),\ lumo(a, -1.246). \end{aligned}$$

where tc stands for the target concept (*active* or *inactive*) satisfied by a.

Literal $atom(a, a_1, carbon, 22, -0.138)$ states that in compound a, atom a_1 is a carbon, of type 22, with partial charge -0.138. Literal $bond(a, a_1, a_2, 7)$ expresses that there exists a (unique) bond between atoms a_1 and a_2 in a, the type of which is 7. This problem typically involves numerical and relational features.

2.2 Overview

DiVS basically combines the Version Space framework and the divide-and-conquer strategy [16]. Examples are generalized one at a time; and the star *Th(Ex)* generalizing the seed Ex is the version space covering Ex, that is, the set of all hypotheses covering Ex and rejecting all examples $Ce_1, \ldots Ce_n$ that do not belong to the same target concept as Ex, called *counter-examples* to Ex[1].
The elementary step in the construction of star *Th(Ex)* consists in building the set $D(Ex, Ce)$ of hypotheses that cover Ex and discriminate a counter-example Ce: *Th(Ex)* is defined as the conjunction of $D(Ex, Ce)$ for Ce ranging over the counter-examples to Ex.

[1] The counter-examples to a positive example Ex are the negative examples, and vice versa.

The overall theory *Th* built by *DiVS* is the disjunction of the version spaces *Th(Ex)* for Ex ranging over the training set.

Disjunctive Version Space Algorithm

```
Th = false.
For each Ex training example
     Th(Ex) = True
     For each Ce counter-example to Ex
          Build D(Ex, Ce)                        (section 2.3,2.5)
          Th(Ex) = Th(Ex) ∧ D(Ex, Ce)
     Th = Th ∨ Th(Ex).
```

DiVS differs from other divide-and-conquer algorithms in one main respect. For most authors [16, 20], seeds are selected among positive examples only: e.g. the different versions of AQ, and *PROGOL* as well, only learn the target concept. In contrast, *DiVS* generalizes positive and negative examples; it thereby learns both the target concept and its negation. This hopefully allows the effects of noisy positive and negative examples to counterbalance each other.

Another key difference between *DiVS* and all other learners, as far as we know, is that *DiVS* does not set any restriction on the number of candidate solutions: it retains all hypotheses partially complete (covering at least one seed) and consistent. In opposition, *FOIL*, *FOCL* and *PROGOL*, among others, aim at finding "the" best hypothesis covering a training example, according to the more or less greedy optimization of a numerical criterion (quantity of information for *FOIL* and *FOCL*, MDL principle for *PROGOL*). To a lesser extent, *ML-Smart* [1, 3] and *REGAL* [7] also look for concise theories.

Let us focus now on building the set $D(Ex, Ce)$ of hypotheses generalizing Ex and rejecting Ce, depending on the hypothesis language.

2.3 Attribute-value learning

In an attribute-value language, the construction of $D(Ex, Ce)$ is straightforward. Consider for instance the positive and negative atoms given in Table 1:

Table 1: Seed and Counter-example

	element	*type*	*electric charge*	*Concept*
Ex	*carbon*	*22*	*3.38*	*positive*
Ce	*hydrogen*	*3*	*-.33*	*negative*

Let hypotheses be conjunctions of selectors ($att = V$) [16], where V denotes an interval in case *att* is linear (e.g. the *type* or *electric charge* of atoms), and a value otherwise.
The set of hypotheses $D(Ex, Ce)$ is given by the disjunction of maximally discriminant selectors, i.e. of the maximally general selectors that cover Ex and

reject Ce:

$$D(Ex, Ce) = (element = carbon) \vee (type > 3) \vee (charge > -.33)$$

Star *Th(Ex)*, given by the conjunction of $D(Ex, Ce)$ for Ce ranging over the counter-examples to Ex, is built with linear complexity in the number N of examples and the number P of attributes. And finally the complexity of *DiVS* is in $\mathcal{O}(N^2 \times P)$ [28].

2.4 First-order logic learning

All positive and negative examples of the target concept are represented via Horn clauses. Since there exists no "standard" semantics for the negation in Logic Programming and even less in Constraint Logic Programming, we explicitly introduce the negation of the target concept *tc*, denoted *opp_tc*: a negative example is a clause the head of which is built on *opp_tc*.

Consider two examples of molecules satisfying the opposite target concepts *active* and *inactive*:

Ex : active(ex) :- atom(ex, a, carbon, 3.38), atom(ex, b, carbon, 1.24).

Ce : inactive(ce) :- atom(ce, c, hydrogen, -.33), atom(ce, d, carbon, 2.16).

We first decompose the seed Ex into a clause $\mathcal{C}$ and a substitution θ, respectively the most general clause and the most specific substitution such that

$$Ex = \mathcal{C}\theta \tag{1}$$

In our toy example, $\mathcal{C}$ stores the structural information of Ex, i.e., that Ex is an *active* molecule having two atoms:

$$\mathcal{C} : active(X) : -atom(X', Y, Z, T), atom(X'', U, V, W)$$

and θ carries all other information in Ex:

$$\theta = \{X/ex,\ X'/ex,\ X''/ex,\ Y/a,\ Z/carbon,\ T/3.38,\ U/b,\ V/carbon,\ W/1.24\}$$

This decomposition allows induction to simultaneously explore two search spaces:

- The space of definite clauses generalizing $\mathcal{C}$. Exploring this space is fairly simple: all variables in $\mathcal{C}$ must be distinct for $\mathcal{C}$ to be the most general clause satisfying equation (1); hence, the only way to generalize $\mathcal{C}$ is by dropping literals.
- A space $\mathcal{F}$ of logical functions on the variables of $\mathcal{C}$. Depending on whether $\mathcal{F}$ is the set of substitutions on $\mathcal{C}$, or the set of constraints on $\mathcal{C}$, *DiVS* either constructs definite clauses [29] or constrained clauses [26, 30]. Note that substitutions on $\mathcal{C}$ are particular cases[2] of constraints on $\mathcal{C}$ [8].
  In this paper, $\mathcal{F}$ is restricted to a subset of constraints on $\mathcal{C}$, to be made more precise later on.

[2] Variable grounding amounts to domain constraint ($X = X.\theta$), where $X.\theta$ denotes the constant X is substituted by according to θ; similarly, variable linking amounts to binary constraint ($X = Y$).

Building the set $D(Ex, Ce)$ of hypotheses that cover Ex and reject Ce amounts here to finding out all pairs (C, ρ), where C generalizes $\mathcal{C}$ (e.g. describes a molecule satisfying the same target concept as Ex and including at most the same number of atoms and bonds) and ρ is a constraint on C that generalizes θ. Furthermore, C and ρ must be such that $C\rho$ discriminates Ce.

In the general case, discrimination can be based on predicates: if C involves a predicate that does not appear in Ce, C discriminates Ce. Predicate-based discrimination is not considered in the following: it does not apply for the considered description of the mutagenesis problem since all molecules involve the same predicates (*atom*, *bond*,..). Besides, it presents no difficulty and can be formalized as a boolean discrimination problem [30].

Constraint-based discrimination takes place when the body of $\mathcal{C}$ (or of the current hypothesis) generalizes that of Ce. Then there exists at least one substitution σ such that $body(\mathcal{C}).\sigma \subseteq body(Ce)$. We then say that $\mathcal{C}$ *matches the negative example* and σ is called *negative substitution*. For instance, in our previous example, negative substitution σ respectively maps the first and second atoms in $\mathcal{C}$ onto the first and second atoms in Ce:

$$\sigma = \{X/ce,\ X'/ce,\ X''/ce,\ Y/c,\ Z/hydrogen,\ T/-.33,\ U/d,\ V/carbon,\ W/2.16\}$$

Whenever a negative substitution exists, $\mathcal{C}$ is inconsistent: its body generalizes the bodies of both Ex and Ce, which yet satisfy opposite target concepts. Constraint-based discrimination prevents such inconsistencies by specializing $\mathcal{C}$: it adds constraints to the body of $\mathcal{C}$ such that negative substitution σ does not satisfy these constraints. For instance, constraint

$$\rho = (Z = carbon)$$

is incompatible with σ, since $Z.\sigma = hydrogen$. By the way, ρ must also generalize the substitution θ derived from Ex, in order for $\mathcal{C}\rho$ to still generalize Ex; e.g. $\rho' = (Z = oxygen)$ is also incompatible with σ, but $\mathcal{C}\rho'$ does not generalize Ex.
A formal presentation of constraint entailment and generalization order will be found in [8]; roughly, constraint ρ_1 generalizes ρ_2 (equivalently, ρ_2 *entails* ρ_1) iff all substitutions satisfying ρ_2 also satisfy ρ_1.

Note that building constraints that generalize θ and are incompatible with a negative substitution σ amounts to an attribute value discrimination problem. This is particularly clear if we restrict our language of constraints to domain constraints, of the form $(X = \mathcal{V})$, where $\mathcal{V}$ is a subset of the domain of X (see section 2.5). This is also true when binary logical and arithmetic constraints are considered (e.g. $(X \neq Y)$, $(Z < T + 10)$, $(S > U - 20)$), by introducing auxiliary variables (this point is detailed in [30]). However, binary constraints will not be further considered here, for two reasons. First of all, introducing binary constraints does not significantly modify the complexity of induction (it only affects its polynomial part), which is our primary concern in this paper. Second, unary constraints turned out to be sufficient to reach a good level of predictive accuracy on the mutagenesis problem.

Finally, our language of constraints is restricted to unary constraints of the form $(X = \mathcal{V})$, where

- $\mathcal{V}$ is an interval if X is a real or integer-valued variable;
- $\mathcal{V}$ is a value if X is a nominal variable.

In particular, if all variables are considered nominal, the language of hypotheses is a restriction of that of logical clauses (only grounding-based specialization applies).

2.5 Characterizing $D(Ex, Ce)$

Let $Ex = \mathcal{C}\theta$ be the seed and let Ce be a counter-example to the seed. Let σ be a negative substitution on $\mathcal{C}$ derived from Ce and let us first assume that σ is the only negative substitution derived from Ce.
Building a maximally discriminant domain constraint $\rho_{X,\sigma}$ on variable X that generalizes θ and is incompatible with σ amounts to building a maximally discriminant selector in the attribute-value case (section 2.3). Constraint ρ_σ is defined as the disjunction of maximally discriminant constraints $\rho_{X,\sigma}$, for X ranging over discriminant variables (e.g. V is not discriminant since $V.\theta = V.\sigma$). In our toy example, variables X, X', $X"$, Y and U, which respectively identify the molecule and the atoms, are not considered as they are irrelevant for discrimination purposes:

	X	X'	Y	Z	T	$X"$	U	V	W
θ	*ex*	*ex*	*a*	*carbon*	3.38	*ex*	*b*	*carbon*	1.24
σ	*ce*	*ce*	*c*	*hydrogen*	$-.33$	*ce*	*d*	*carbon*	2.16

$$\rho_\sigma = (Z = carbon) \vee (T > -.33) \vee (W < 2.16)$$

It is shown that any constraint generalizing θ discriminates σ iff it entails (is generalized by) ρ_σ [30]. A clause $C\rho$ therefore belongs to $D(Ex, Ce)$ iff $C\rho$ generalizes Ex and ρ entails ρ_σ.

In the general case, let $\Sigma_{Ex,\, Ce}$ be the set of negative substitutions on $\mathcal{C}$ derived from Ce. A constraint ρ must be incompatible with *all* negative substitutions derived from Ce, in order for $\mathcal{C}\rho$ to be consistent with Ce. $D(Ex, Ce)$ can thus be characterized as follows [30]:

Proposition 1. $C\rho$ belongs to $D(Ex, Ce)$ iff $C\rho$ generalizes Ex and ρ entails ρ_σ for all σ in $\Sigma_{Ex,\, Ce}$.

To sum up, $D(Ex, Ce)$ is computationally described by $\mathcal{C}$ and the set of constraints $\{\rho_\sigma \; s.t. \; \sigma \in \Sigma_{Ex,\, Ce}\}$.

This characterization can be used to reach the two main goals of machine learning: that of explicitly characterizing the constructed theory and that of classifying further instances of the problem domain. Some results and a method addressing the first aim of learning were presented in our previous work [30]. We therefore concentrate here on the second aim of learning, that is, classification.

2.6 Classifying further examples

As a matter of fact, computational descriptions as above are sufficient to classify unseen instances of the problem domain, via a nearest neighbor-like decision process:

• An unseen instance E is termed *neighbor* of a training example E iff E belongs to *Th(Ex)*, that is, is covered by a hypothesis in *Th(Ex)*.

• By construction, E belongs to *Th(Ex)* iff E belongs to all $D(Ex, Ce)$, for Ce ranging over the counter-examples to Ex.

• The computational characterization of $D(Ex, Ce)$ is sufficient to check whether an unseen instance E belongs to $D(Ex, Ce)$ [30]:

Proposition 2. E belongs to $D(Ex, Ce)$ iff E can be expressed as $C\tau$, where C generalizes $\mathcal{C}$ and τ entails ρ_σ for all σ in $\Sigma_{Ex,\ Ce}$.

Let $\Sigma_{Ex,\ E}$ denote the set of substitutions on $\mathcal{C}$ matching E. Then, proposition 2 is translated as:

Belongs(E, $D(Ex, Ce)$)

```
    For each τ in Σ_{Ex, E}
        If τ entails ρ_σ for all σ in Σ_{Ex, Ce},
            return true.
    return false.
```

And

Neighbor (E, Ex) : *(E belongs to Th(Ex))*

```
    For each Ce counter-example to Ex
        if NOT Belongs(E, D(Ex,Ce))
            return false
    return true.
```

Simply put, our approach constructs an oracle rather than an explicit theory. This oracle is made of theory *Th*, stored as the list of

$$D(Ex_i, Ex_j) = (\mathcal{C}_i, \{\rho_\sigma \ s.t.\ \sigma \in \Sigma_{Ex_i, Ex_j}\})$$

for Ex_i and Ex_j training examples satisfying different target concepts. Theory *Th*, interpreted according to Proposition 2, allows one to compute the boolean *Neighbor(E, Ex)* function, and this function together with a standard nearest neighbor algorithm, achieves the classification of any further instance E.

This approach can be compared to that of *RIBL* [5] which is also based on nearest neighbors. The essential difference is the following: in *RIBL*, the similarity between E and a training example Ex only depends on E and Ex (this is true also for the even more sophisticated first-order similarity used in KBG [2]). But here, the neighborhood of Ex (and the fact that E is neighbor of Ex or not) depends on E, Ex *and* the counter-examples $Ce_1, \ldots Ce_n$ to Ex: the underlying similarity is driven by discrimination.

2.7 Complexity

Under the standard assumption that the domain of any variable is explored with a bounded cost, the complexity of building ρ_σ is linear in the number of variables in $\mathcal{C}$ (it would be quadratic if binary constraints were also considered). Let $\mathcal{X}$ and $\mathcal{S}$ respectively denote upper-bounds on the number of variables in $\mathcal{C}$ and on the number of substitutions in Σ_{Ex_i,Ex_j}. The characterization of $D(Ex_i, Ex_j)$ is then in $\mathcal{O}(\mathcal{X} \times \mathcal{S})$.

Let N be the number of training examples. Since all $D(Ex_i, Ex_j)$ must be characterized, the complexity of learning in *DiVS* is

$$\mathcal{O}(N^2 \times \mathcal{X} \times \mathcal{S})$$

And, since checking whether an instance E belongs to *Th(Ex)* requires to consider all substitutions in $\Sigma_{Ex,\ E}$, the number of which is upper-bounded by $\mathcal{S}$, the complexity of classification in *DiVS* is

$$\mathcal{O}(N^2 \times \mathcal{X} \times \mathcal{S}^2)$$

The crux of complexity lies in factor $\mathcal{S}$, which is exponential in the number of literals built on a predicate symbol in the examples [29]. This shows up in the mutagenesis problem, as the number of atoms in a molecule ranges up to 40. $\mathcal{S}$ thus is 40^{40}...

3 Polynomial Approximate Learning

The presented approach suffers from two major drawbacks: first, it is intractable for truly relational problems. Second, inasmuch it stems from the Version Space framework, it is ill-prepared to deal with noisy and sparse data.

The tractability limitation is first addressed via a stochastic bias: the idea consists in sampling, rather than exhaustively exploring, the set of substitutions $\Sigma_{Ex,Ce}$. We again illustrate the stochastic sampling mechanism on the mutagenesis problem.

Second, two heuristics, taken from the propositional version of *DiVS* [28], are used to relax the standard consistency and generality requirements of Version Spaces, and cope with noise and sparseness.

3.1 Stochastic Bias

Let us have a closer look at the negative substitutions explored by *DiVS*.

In the mutagenesis problem, the semantics of a molecule is not modified by changing the identifiers of the atoms (nominal constants $a_1, a_2, \ldots a_i$). These identifiers can thus be arbitrarily set to $1, 2, \ldots, n$, if n denotes the number of atoms in Ex. A negative substitution σ on $\mathcal{C}$ is completely defined by associating each atom in $\mathcal{C}$, which corresponds to a given atom i in Ex w.r.t. θ, to an atom in Ce denoted $\sigma(i)$ by abuse of notation. The intractability of *DiVS* follows from the fact that the number of such substitutions is in ${n'}^{\,n}$, if n' denotes the number of atoms in Ce.

Let us concentrate on atoms for the sake of readability.
Discriminating σ from θ requires to discriminate at least one atom i in Ex from atom $\sigma(i)$ in Ce. The more "similar" atoms i in Ex and $\sigma(i)$ in Ce, the more difficult it is to discriminate them, and the more informative the negative substitution σ is: this notion parallels that of near-misses in attribute-value languages. Formally, a partial order can be defined on the substitutions in $\Sigma_{Ex,Ce}$, and it is shown that non-minimal substitutions can soundly be pruned with regards to discriminant induction [26, 29]: this pruning is analogous to the pruning of non near-misses examples in the propositional case [32, 27]. Unfortunately, building the set of such minimal substitutions turns out to be intractable too.

Another possibility would be to consider *the best* substitution σ, defined as minimizing some distance to θ in the line of the structural similarity developed in [2]. For instance, the best substitution in $\Sigma_{Ex,Ce}$ would minimize the sum of the distances between atom i in Ex and atom $\sigma(i)$ in Ce.
As noted in [33], the description of an atom can be handled as a single tree-structured feature since the element of an atom commands its atom type (e.g. the atom type of a *hydrogen* atom is in [1,3] whereas the atom type of a carbon atom is in [21,24]) and the atom type similarly commands its electric charge. Defining a distance between any two atoms thus is straightforward: the distance of two atoms having same atom type is the difference of their electric charges; otherwise, if the atoms are of the same element, their distance is the difference of their atom type, augmented by a sufficiently large constant (twice the maximal electric charge); otherwise (the atoms are of different elements), their distance is set to another constant (twice the maximal electric charge plus the maximal atom type).

However, using an optimization approach to determine which substitution to consider in $\Sigma_{Ex,Ce}$ raises several problems: first of all, we feel that a single substitution, even optimal, cannot be representative of the whole set $\Sigma_{Ex,Ce}$; second, this combinatorial optimization problem is itself computationally expensive...

Finally, we decided to consider several substitutions, the number of which to be supplied by the user.
These substitutions could have been purely randomly defined, except that, as stated above, substitutions nearer to θ are more informative. When constructing a substitution σ, one thus associates to any atom i in Ex the atom j in Ce which is most similar to i, provided that j is not yet associated to another atom in Ex: atom j in Ce has same electric charge as atom i in Ex, if possible; otherwise, it has same atom type; otherwise, it is of same element.
Let n and n' respectively denote the number of atoms in Ex and Ce. The sampling mechanism of the substitutions in $\Sigma_{Ex,\ Ce}$ is currently implemented as follows:

Select σ in $\Sigma_{Ex,\ Ce}$

```
while possible
    Select i in {1,...,n} not yet selected
        Select j in {1,...,n'} not yet selected such
        that atom j in Ce is as close as possible
        to atom i in Ex,
            Do σ(i) = j.
```

Note that index j is deterministically selected depending on i, and i is stochastically selected with uniform probability in $\{1, \ldots, n\}$. This way, any atom i in Ex will in average be associated to a similar atom in Ce, provided the sampling mechanism is run a sufficient number of times.

More precisely, the above stochastic sampling mechanism ensures that a set of samples captures an arbitrarily precise representation of $\Sigma_{Ex,\ Ce}$ with high probability, provided the number of samples allowed is "sufficient". Further work is concerned with formalizing this intuition, as well as improving the selection mechanism via taking into account also the bonds between atoms.

3.2 Overview of *STILL*

The *STILL* algorithm combines the general approach of *DiVS* and the above sampling mechanism. This stochastic bias is used to make both induction and classification tractable.

Approximate Learning Remember that *DiVS* constructs the set *Th(Ex)* of consistent hypotheses that cover Ex, through exploring the whole sets of substitutions $\Sigma_{Ex,\ Ce}$ for Ce ranging over the counter-examples to Ex. Instead of that, *STILL* only processes η substitutions, where η is a positive integer supplied by the user. This way, it constructs a set of hypotheses $Th_\eta(Ex)$ that cover Ex and are only partially ensured to be consistent, since only sampled substitutions are ensured to be discriminated.

Concretely, the set of hypotheses $Th_\eta(Ex)$ is characterized by clause $\mathcal{C}$ (with $Ex = \mathcal{C}\theta$) and a set of constraints $\mathcal{R}$, including η discriminant constraints built

as follows. Let n be the number of counter-examples to Ex; for each counter-example Ce, $\frac{\eta}{n}$ samples of substitutions are selected in $\Sigma_{Ex,\ Ce}$. $\mathcal{R}$ is composed of the constraints ρ_σ discriminating the selected samples of substitutions derived from all counter-examples.
This heuristics ensures that the specificity of star $Th_\eta(Ex)$ does not depend on whether the seed Ex belongs to the minority or the majority class (this would not be the case if the number of constraints in $\mathcal{R}$ were proportional to the number of counter-examples to Ex).
It was adopted for reasons of empirical accuracy, as examples in the mutagenesis application are distributed two active to one inactive.

Characterize $Th_\eta(Ex)$:

$\mathcal{R} = \phi$.
n = `Number of counter-examples to` Ex
`For` Ce `counter-example to` Ex
  `For` $j = 1 \ldots \frac{\eta}{n}$,
    `Select` σ `in` $\Sigma_{Ex,\ Ce}$,
    `Build` ρ_σ
    `Do` $\mathcal{R} = \mathcal{R} \bigcup \{ \rho_\sigma \}$
`return` $(\mathcal{C}, \mathcal{R})$.

The disjunction Th_η of theories $Th_\eta(Ex)$ for Ex ranging over the training set, is termed *approximate theory*; the number η of allowed samples is termed *rate of approximation*. Note that Th_η is more general than Th and tends toward Th as η increases.

Approximate classification. The classification process in *DiVS* is based on checking which training examples are neighbors of the instance E to classify (section 2.6). In order to check whether E is neighbor of Ex, i.e. belongs to *Th(Ex)*, *DiVS* explores the set $\Sigma_{Ex,\ E}$ of substitutions on $\mathcal{C}$ (where $Ex = \mathcal{C}\theta$), matching E. The size of this set similarly makes classification intractable.

STILL again addresses this limitation via the sampling mechanism: instead of exhaustively exploring $\Sigma_{Ex,\ E}$, it only considers K substitutions in this set, where K is a positive integer supplied by the user. E is termed *approximate neighbor* of Ex if at least one in K samples of $\Sigma_{Ex,\ E}$ entails all constraints in *Th(Ex)*:

Approx_Neighbor (E, Ex) :

$(\mathcal{C}, \mathcal{R})$ = `Characterize` $Th_\eta(Ex)$
`For i = 1...K`
  `Select` τ `in` $\Sigma_{Ex,E}$
    `If` τ `entails all` ρ `in` $\mathcal{R}$
      `return` *true*
`return` *false*

The classification in *STILL* is finally done according to the standard nearest neighbor algorithm, based on the above *Approx_Neighbor* function.

Note the above function corresponds to an "interpretation" of $Th_\eta(Ex)$ that is more specific than $Th_\eta(Ex)$ itself; this over-specificity decreases as K increases.

Parameter K controls the number of trials allowed to get an answer from theory Th_η; metaphorically speaking, K corresponds to the "patience" of the constructed expert.

3.3 Coping with noisy and sparse examples

$Th(Ex)$ (which is the theory $Th_\eta(Ex)$ tends toward as η increases) includes consistent hypotheses only, and maximally general consistent hypotheses in particular. No doubt this approach is ill-suited to real-world datasets: when erroneous examples are encountered, strictly consistent hypotheses have few predictive accuracy [4]. And when examples are sparse, maximally general consistent hypotheses are too general: most instances come to be covered by a hypothesis in most $Th_\eta(Ex_i)$, and therefore get unclassified, or classified in the majority class.

These limitations were already encountered in the attribute-value version of *DiVS*, and have been addressed by two heuristics [28], which simply extend to first-order logic owing to the computational characterization of the constructed theory.

The presence of noise in the data is classically addressed by relaxing the consistency requirement. This is done at classification time, via modifying the test of neighborhood. By definition, E is considered as neighbor of Ex iff it belongs to $D(Ex, Ce)$ for all Ce counter-example to Ex. This definition is simply relaxed as: E is from now on considered as neighbor of Ex iff it belongs to $D(Ex, Ce)$ for all Ce counter-example to Ex, except at most ε of them, where ε is a positive integer supplied by the user. The greater ε, the wider the neighborhood of Ex is.

The sparseness of the data is addressed by increasing the specificity of the produced theory. This modification also takes place during classification, and regards the test of constraint entailment. By construction, constraint ρ_σ is the maximally general constraint that discriminates σ and generalizes θ; it is the disjunction of domain constraints $\rho_{X,\sigma}$ for X ranging over the variables of $\mathcal{C}$. A given substitution τ hence entails ρ_σ iff there exists at least one variable X such that $X.\tau$ satisfies $\rho_{X,\sigma}$.

The specificity of the theory is tuned by considering from now on that τ entails ρ_σ iff τ satisfies at least M domain constraints $\rho_{X,\sigma}$ (instead of one), where M is a positive integer supplied by the user. This amounts to considering ρ_σ as an $M - of - N$ concept. The greater M, the smaller the neighborhood of Ex is.

Note that the constructed theory does not depend in any way on the values of parameters ε or M. In particular, *STILL* requires no *a priori* knowledge regarding the rate of noise and representativity of the data. Parameters M and ε can be adjusted from the experimental classification results — but with no need to restart induction. See [28] for a discussion about the advantages of such *a posteriori* biases.

3.4 Complexity

As expected, the stochastic bias cuts down the complexity of learning and classification.

Let $\mathcal{X}$ still denote an upper-bound on the number of variables in $\mathcal{C}$. The complexity of building ρ_σ is still linear in $\mathcal{X}$. The construction of one sample σ is quadratic in $\mathcal{X}$ (this is a large over-estimation). Hence, the complexity of learning $Th_\eta(Ex)$ is in $\mathcal{O}(\mathcal{X}^3 \times \eta)$. Finally, the computational complexity of induction in *STILL* is linear in the rate of approximation and in the number of training examples, and cubic in the number of variables in one example:

$$\mathcal{O}(N \times \mathcal{X}^3 \times \eta)$$

In the mutagenicity problem, N is 188, $\mathcal{X}$ is less than 200. The rate of approximation η was set to 300, to be compared with the typical size of a set $\Sigma_{Ex,\ Ce}$, that is 30^{30}.

The complexity of classification is that of induction increased by factor K, which was set to 3 in our experiments:

$$\mathcal{O}(N \times \mathcal{X}^3 \times \eta \times K)$$

Note that the heuristics designed to cope with noise and sparseness do not modify the computational complexity of classification.

4 Experimentation

This section presents an experimental validation of the algorithms described above on the well-studied mutagenicity problem (see [33] for a detailed presentation of this problem).

4.1 The data

The dataset is composed of 125 active molecules and 63 inactive molecules. Four levels of description of these molecules have been considered in the literature [33, 10]: Background knowledge $\mathcal{B}_1$ includes the description of atoms and bonds in the molecules. Background knowledge $\mathcal{B}_2$ stands for $\mathcal{B}_1$ augmented with definition of numeric inequalities. Background knowledge $\mathcal{B}_3$ is $\mathcal{B}_2$ augmented with a non structural description of the molecules (five numerical and boolean attributes). Background knowledge $\mathcal{B}_4$ stands for $\mathcal{B}_3$ augmented with the definition of simple chemical concepts (e.g. benzenic or methyl group).

The reference results obtained by *PROGOL* and *FOIL* on this problem (reported from [33] and [34]), are given in Table 1. The run times (in seconds) are measured on HP-735 workstations.

Background knowledge	Accuracy		Time	
	FOIL	*PROGOL*	*FOIL*	*PROGOL*
$\mathcal{B}_1$	60 ± 4	76 ± 3	5 000	117 000
$\mathcal{B}_2$	81 ± 3	81 ± 3	9 000	64 350
$\mathcal{B}_3$	83 ± 3	83 ± 3	.5	42 120
$\mathcal{B}_4$	82 ± 3	88 ± 2	.5	40 950

Table 1: Results of FOIL *and* PROGOL *on the 188-compound problem: Average predictive accuracy on the test set*

In this paper, all experiments conducted with *STILL* only consider background knowledge $\mathcal{B}_3$. The 11 264 ground facts composing $\mathcal{B}_3$ are partitioned in 188 ground clauses, each clause including all information relevant to a given compound.

4.2 Experimental Aim

Previous experiments with *STILL* conducted with background knowledge $\mathcal{B}_2$ [31] have shown the validity of this approach in terms of predictive accuracy. Nevertheless, the reason why *STILL* obtains such good results is still unclear.

A first explanation is related to the powerful formalism of constraint logic programming, and the use of inequality constraints relative to either the electric charge or type of element of the atoms, or the non structural description part of molecules (attributes *logp* and *lumo*). The use of numerical inequalities is typically responsible for the increase of performance of *FOIL* from $\mathcal{B}_1$ to $\mathcal{B}_2$ (Table 1).

The influence of the hypothesis language is evaluated with two experiments. In the first one, *STILL* handles all information in the examples as if it were nominal; only constraints such as $(X = X.\theta)$ are learned, which means that *STILL* constructs pure definite clauses. In the second experiment, inequality constraints can be set on numerical variables and *STILL* thus constructs constrained clauses.

A second explanation is related to the redundancy of the constructed theory. It has been suggested that redundant classifiers tend to be more robust and reliable than concise ones [6, 23]. *STILL* involves two kinds of redundancy. First, it both constructs the theory of mutagenic activity and that of inactivity; in opposition, *PROGOL* only constructs the theory of activity. Second, *STILL* generalizes all examples, whereas both *PROGOL* and *FOIL* remove the examples covered by previous hypotheses.

To check to what extent redundancy is a key factor of accuracy in our approach, *STILL* is compared to a variant denoted *AQ-STILL*, which does include some selection of the seeds. More precisely, *AQ-STILL* only generalizes those examples which are not yet correctly classified at the time they are considered.

In summary, four variants of *STILL* are implemented[3] and compared:

- $STILL^{CLP}$, which corresponds to the approach described throughout this paper, where all examples are generalized and inequality constraints can be set on numerical variables;
- $STILL^{ILP}$, where all examples are generalized but specialization is limited to variable grounding;
- $AQ\text{-}STILL^{CLP}$, which differs from $STILL^{CLP}$ in the selection of seeds; and
- $AQ\text{-}STILL^{ILP}$, which similarly differs from $STILL^{ILP}$ in the selection of seeds.

4.3 Experimental Settings

The parameters controlling the stochastic biases are constant in the following experiments:
The rate of approximation η, that ensures the tractability of induction, is set to 300.
The parameter K, that ensures the tractability of classification, is set to 3.

Parameter M used to control the specificity of the theory varies from 1 to 10. Parameter ε, which corresponds to the maximal number of inconsistencies of a hypothesis, varies from 0 to 4.

All results are averaged over 15 independent runs, where each run consists in learning from 90% of the 188 compounds (randomly selected such that the ratio of active/inactive compounds in the training set is same as in the global data, i.e. about two to one) and classifying the remaining 10% of the data. This protocol of validation is similar to the ten-fold cross validation used in [33]; the number of runs is only slightly increased (from 10 to 15), as suggested for stochastic approaches [9].

For each setting of parameters ε and M, the average percentage of test examples correctly classified, unclassified[4] and misclassified are indicated (labels *Accur*, ? and *Misclass*), together with the standard deviation of the accuracy (label $\pm$). The average run time on HP-735 workstations, including the construction of the theory and the classification of the test examples, is also given (in seconds).
Last, the average number of seeds is also indicated in the case of $AQ\text{-}STILL^{CLP}$ and $AQ\text{-}STILL^{ILP}$; in the case of $STILL^{CLP}$ and $STILL^{CLP}$, the number of seeds exactly is the number of training examples (170), and has been omitted.

4.4 CLP versus ILP

Table 2 shows the results obtained by $STILL^{CLP}$ and $STILL^{ILP}$. Remember the only difference lies in the possibility for $STILL^{CLP}$ to set inequality constraints

[3] In C^{++}.

[4] An example gets unclassified if either it admits no neighbor in the seeds, or if the majority vote ends up in a tie.

on the type of atom and electric charge of atoms, and on the numerical attributes *logp* and *lumo*.

As noted by [12], one cannot conclude from the presence of numbers in a problem, that this problem needs a learner with numerical skills: many learning problems with numbers include in fact very few distinct values, and can therefore be handled by purely symbolic means. But Table 2 witnesses that the mutagenesis problem does benefit from a CLP formalism.

ε	$STILL^{ILP}$						$STILL^{CLP}$					
	M	Accur.	?	Miscl.	±	Time	M	Accur.	?	Miscl.	±	Time
0	1	86.7	0.74	12.6	± 6.9	51	4	88.5	0.37	11.1	± 6.1	73
	2	83.3	3.3	13.3	± 7.7	57	5	91.1	0	8.89	± 7.3	78
	3	**86.7**	3	10.4	± 8.1	59	6	90.7	1.1	8.15	± 4.6	82
	4	81.1	9.6	9.26	± 5.4	60	7	**91.5**	1.1	7.41	± 3.8	86
	5	77.4	14	8.52	± 11	60	8	86.7	3.7	9.63	± 8.1	90
	6	68.9	19	12.2	± 12	59	9	91.1	2.2	6.67	± 5.8	93
	7	64.4	26	9.63	± 7.7	58	10	85.2	4.1	10.7	± 7.3	97
2	1	83.3	0.74	15.9	± 7.7	49	4	89.3	1.3	9.4	± 7.7	69
	2	83.7	1.1	15.2	± 4.6	55	5	88.9	0	11.1	± 7.3	74
	3	**83.7**	1.9	14.4	± 12	58	6	**92.5**	0.4	7.14	± 5	77
	4	80.4	3.7	15.9	± 6.5	60	7	91.5	0	8.52	± 5.4	81
	5	79.6	4.8	15.6	± 7.3	59	8	91.9	0.37	7.78	± 5.4	85
	6	80.7	4.8	14.4	± 10	59	9	87.4	0.37	12.2	± 6.1	88
	7	75.2	9.6	15.2	± 6.5	58	10	87.4	0.74	11.9	± 6.5	92
4	1	82.6	0.74	16.7	± 8.9	47	4	81.1	0	18.9	± 5	66
	2	**85.6**	1.1	13.3	± 6.1	53	5	85.6	1.5	13	± 8.5	71
	3	79.6	1.9	18.5	± 11	57	6	85.9	0.37	13.7	± 8.5	75
	4	78.5	2.6	18.9	± 7.3	59	7	**89.3**	0.37	10.4	± 8.9	78
	5	81.1	3	15.9	± 14	59	8	88.9	1.5	9.63	± 6.6	82
	6	77	5.6	17.4	± 8.1	59	9	84.8	1.1	14.1	± 6.1	85
	7	75.6	11	13	± 14	58	10	85.9	0.37	13.7	± 7.7	89

(a) Without inequality constraints (b) With inequality constraints

Table 2: Results of STILL *on* $\mathcal{B}_3$, $\eta = 300$, $K = 3$

Note that the optimal accuracy for $STILL^{CLP}$ corresponds to higher values of M (M = 6-7), than for $STILL^{ILP}$ (M = 2-3). This could be explained as follows. Hypotheses constructed by $STILL^{CLP}$ are more general than those constructed by $STILL^{ILP}$, for numerical inequalities are more often satisfied than equalities. But the specificity of hypotheses also increases with parameter M, which offsets the extra generality permitted by the CLP language.

The most striking fact is that the best result of $STILL^{CLP}$ (accuracy 92.5% ± 5 for $\varepsilon = 2$ and $M = 6$) outperforms that of *PROGOL* (Table 1) and *FORS* (with accuracy 89% ± 6) [10], despite the fact that *FORS* explores a language of hypotheses (including linear expressions of the numerical variables) much richer than that of $STILL^{CLP}$.

Of course, a fair comparison would require to see how the predictive accuracy of other learners varies with their control parameters; e.g. [33] only indicates the results obtained with *PROGOL* for a maximal number of inconsistencies set to 4, and a maximal number of literals in a clause set to 5.

This asks the question of how to automatically adjust the parameters of *STILL*. On-going experiments are concerned with using the training set to tune M and ε in the line of [13].

4.5 Pruning the seeds: Pros and Cons

Table 3 shows the results obtained by $AQ\text{-}STILL^{CLP}$ and $AQ\text{-}STILL^{ILP}$. Remember that the only difference between *STILL* and *AQ-STILL* lies in the selection of the seeds: *STILL* generalizes all examples whereas *AQ-STILL* only generalizes examples that are not correctly classified at the time they are considered.

ε	$AQ\text{-}STILL^{ILP}$							$AQ\text{-}STILL^{CLP}$						
	M	Accur.	?	Miscl.	±	Time	Seeds	M	Accur.	?	Miscl.	±	Time	Seeds
	1	**83.3**	7	9.63	± 11	59	36.3	2	79.3	12	8.52	± 10	27	21.4
	2	82.6	7	10.4	± 8.9	86	43.9	3	**87.8**	5.6	6.67	± 7.3	30	17.6
	3	79.3	10	10.4	± 11	117	54.9	4	81.9	9.3	8.89	± 6.9	39	17.6
0	4	74.1	17	8.52	± 7.3	139	65.3	5	81.9	8.9	9.26	± 9.6	48	18.7
	5	77	17	5.93	± 8.9	161	77.5	6	80.7	9.3	10	± 8.1	65	23.7
	6	68.1	26	6.3	± 10	179	90.4	7	82.6	11	5.93	± 9.2	89	29.5
	7	60.4	34	5.19	± 14	189	99.3	8	84.1	10	5.56	± 7.3	113	34.8
	1	**80**	8.5	11.5	± 10	54	39.5	3	84.8	6.3	8.89	± 8.5	42	32
	2	78.5	7.8	13.7	± 12	74	41.2	4	79.3	7	13.7	± 6.5	44	26.8
	3	74.8	14	10.7	± 9.2	95	47	5	84.4	7.4	8.15	± 9.6	50	25.5
2	4	75.2	12	13	± 9.2	120	57.9	6	84.4	8.1	7.41	± 7.7	62	27.2
	5	69.6	17	13.3	± 12	133	64.6	7	84.1	8.5	7.41	± 8.1	70	27.4
	6	69.6	20	10.7	± 10	147	73.5	8	**86.3**	5.9	7.78	± 10	82	31.4
	7	72.2	20	7.41	± 11	156	82.3	9	80.7	6.7	12.6	± 7.3	95	33.9
	1	82.2	6.7	11.1	± 8.5	49	40.3	4	85.2	5.2	9.63	± 8.5	47	31.8
	2	**83.3**	4.8	11.9	± 6.1	69	42.1	5	84.8	6.3	8.89	± 8.1	51	28.9
	3	81.1	5.9	13	± 11	94	48.8	6	85.2	5.9	8.89	± 8.1	60	28.3
4	4	77.8	9.6	12.6	± 2.4	113	56.5	7	**85.6**	5.6	8.89	± 8.9	67	29.5
	5	72.6	15	12.6	± 7.3	124	61.4	8	83.3	5.6	11.1	± 7.3	76	30.1
	6	70	19	11.1	± 11	138	69.5	9	82.2	4.8	13	± 6.8	88	31.1
	7	64.8	23	11.9	± 9.6	147	77.2	10	81.1	8.5	10.4	± 10	105	36

(a) Without inequality constraints (a) With inequality constraints

Table 3: Results of AQ-STILL *on* $\mathcal{B}_3$, $\eta = 300$, $K = 3$

Again, the use of inequality constraints appears beneficial as $AQ\text{-}STILL^{CLP}$ outperforms $AQ\text{-}STILL^{ILP}$.

In what regards the number of seeds, it increases with M in $AQ\text{-}STILL^{ILP}$: as hypotheses get more and more specific, more and more training examples are unclassified at the time they are considered, and they are therefore generalized.

In $AQ\text{-}STILL^{CLP}$, the same trend is observed for high values of M. The number of seeds also increases for small values of M, but for another reason: when M is small, stars constructed by $AQ\text{-}STILL^{CLP}$ are overly general; more and more examples are therefore misclassified and generalized.

The important fact is that the best predictive accuracy of $AQ\text{-}STILL^{CLP}$ appears lower than that of $STILL^{CLP}$ (87.8 $\pm$ 7 against 92.5 $\pm$ 5). This tends to support our claim that redundancy is a key factor of predictive accuracy [30, 28].

Moreover, the benefit of pruning is unclear with regards to the computational cost: *AQ-STILL* includes the classification of training examples (in order to check whether they can be pruned), which means that the computational complexity of learning is in $\mathcal{O}(N^2 \times \mathcal{X}^3 \times \eta \times K)$, whereas it is $\mathcal{O}(N \times \mathcal{X}^3 \times \eta)$ for *STILL*. Factually, the computational cost strongly depends on the number of seeds: when the number of seeds is high, as it is the case for $AQ\text{-}STILL^{ILP}$, pruning globally hinders learning.

4.6 General remarks and further experiments

As shown in Tables 2 and 3, the run-times of *STILL* range from 50 to 180 seconds (these include the construction of the theory and the classification of the test examples on HP-735 workstations). Similar run-times were obtained with background knowledge $\mathcal{B}_1$ and $\mathcal{B}_2$ [31]. This fully demonstrates the potential of the stochastic bias presented in this paper, to master the combinatorial complexity of induction in first order languages.

However, these good results could be due to the ad hoc sampling mechanism designed for the mutagenicity problem (section 3.1). On-going research is concerned with a pure random sampling mechanism.

The influence of parameters η and K must also be studied. Preliminary experiments with $\eta = 700$ on background knowledge $\mathcal{B}_2$ show the expected increase in the computational cost (linear in η) but only bring a slight improvement of the best predictive accuracy.

5 Conclusion

We have experimentally demonstrated the potential of the stochastic approximate learner *STILL* for classification purposes.

The main interest of this work, in our sense, is to show how stochastic processes can be engineered to cut down the combinatorial complexity pertaining to ILP. Further research is concerned with improving the sampling mechanism (e.g. to take into account also the bonds between atoms), while preserving an acceptable complexity.

Note that this sampling mechanism supports, rather than replaces, induction. This is a strong difference with the genetic side of machine learning and ILP [14, 36]: what is sampled here is related to examples rather than to solutions.

Another interest lies in the non-standard use of the Version Space framework: the computational representation of the constructed theory sidesteps the intrinsic combinatorial complexity of Version Spaces. Further, it allows one to relax at no additional cost the consistency and generality requirements, whenever this is required by the defects, noise and sparseness of the data. Moreover, experiments demonstrate the benefit of learning redundant theories; and Version Spaces are indeed maximally redundant theories.

The main weakness of our learning approach is that it constructs nothing like an intelligible theory. Further work is concerned with pruning and compacting the inarticulate theory underlying the classification process; the challenge lies in providing a readable version of this theory *having same predictive accuracy*. The key question is that of the long debated trade-off between intelligibility and predictive accuracy.

This approach will also be given a learnability model, be it based on PAC-learnability [35] or U-learnability [19]. In particular, in the *Probably Approximately Correct* (PAC) framework, parameter η used to sample the substitutions naively corresponds to the probability of getting the desired theory, whose approximate correction is ϵ.

Acknowledgments. We are grateful to S. Muggleton, A. Srinivasan and R. King, who formalized, studied and made available the mutagenicity problem: this nice problem was determinant for the orientation of the presented work. The work of the authors has been partially supported by the ESPRIT BRA 6020 Inductive Logic Programming and by the ESPRIT LTR 20237 ILP^2.

References

1. F. Bergadano and A. Giordana. Guiding induction with domain theories. In Y. Kodratoff and R.S. Michalski, editors, *Machine Learning : an artificial intelligence approach*, volume 3, pages 474–492. Morgan Kaufmann, 1990.
2. G. Bisson. Learning in FOL with a similarity measure. In *Proceedings of* 10^{th} *AAAI*, 1992.
3. M. Botta and A. Giordana. Smart+ : A multi-strategy learning tool. In *Proceedings of IJCAI-93*, pages 937–943. Morgan Kaufmann, 1993.
4. P. Clark and T. Niblett. Induction in noisy domains. In I. Bratko and N. Lavrac, editors, *Proc. of European WorkShop on Learning*, pages 11–30. Sigma Press, 1987.
5. W. Emde and D. Wettscherek. Relational Instance Based Learning. In L. Saitta, editor, *Proceedings of the* 13^{th} *International Conference on Machine Learning*, pages 122–130, 1996.
6. M. Gams. New measurements highlight the importance of redundant knowledge. In K. Morik, editor, *Proceedings of EWSL-89*, pages 71–80. Pitman, London, 1989.
7. A. Giordana and L. Saitta. REGAL: An integrated system for learning relations using genetic algorithms. In *Proceedings of* 2^{nd} *International Workshop on Multistrategy Learning*, pages 234–249. Harpers Ferry, 1993.
8. J. Jaffar and J. L. Lassez. Constraint Logic Programming. In *Proc. of the fourteenth ACM Symposium on the Principles of Programming Languages*, pages 111–119, 1987.

9. K. E. Kinnear. A perspective on GP. In K. E. Kinnear, editor, *Advances in Genetic Programming*, pages 3–19. MIT Press, Cambridge, MA, 1994.
10. A. Karalic. *First Order Regression.* PhD thesis, Institut Josef Stefan, Ljubljana, Slovenia, 1995.
11. R.D. King, A. Srinivasan, and M.J.E. Sternberg. Relating chemical activity to structure: an examination of ILP successes. *New Gen. Comput.*, 13, 1995.
12. R. Kohavi. The power of decision tables. In N. Lavrac and S. Wrobel, editors, *Proceedings of ECML-95, European Conference on Machine Learning*, pages 174–189. Springer-Verlag, 1995.
13. R. Kohavi and G.H. John. Automatic Parameter Selection by Minimizing Estimated Error. In A. Prieditis and S. Russell, editors, *Proceedings of ICML-95, International Conference on Machine Learning*, pages 304–312. Morgan Kaufmann, 1995.
14. M. Kovacic. MILP: A stochastic approach to ILP. In S. Wrobel, editor, *Proceedings of ILP-94, International Workshop on Inductive Logic Programming*, 1994.
15. N. Lavrac, S. Dzeroski, and M. Grobelnick. Learning non recursive definitions of relations with LINUS. In *Proceedings of EWSL'91*, 1991.
16. R.S. Michalski. A theory and methodology of inductive learning. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning : an artificial intelligence approach*, volume 1. Morgan Kaufmann, 1983.
17. R.S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The AQ15 inductive learning system: an overview and experiment. In *Proceedings of IMAL*, 1986.
18. T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
19. S. Muggleton. Bayesian inductive logic programming. In M. Warmuth, editor, *Proceedings of COLT-94, ACM Conference on Computational Learning*, pages 3–11. ACM Press, 1994.
20. S. Muggleton. Inverse entailment and PROGOL. *New Gen. Comput.*, 13:245–286, 1995.
21. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19:629–679, 1994.
22. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st conference on algorithmic learning theory*. Ohmsha, Tokyo, Japan, 1990.
23. R. Nok and O. Gascuel. On learning decision committees. In A. Prieditis and S. Russell, editors, *Proceedings of ICML-95, International Conference on Machine Learning*, pages 413–420. Morgan Kaufmann, 1995.
24. M. Pazzani and D. Kibler. The role of prior knowledge in inductive learning. *Machine Learning*, 9:54–97, 1992.
25. J.R. Quinlan. Learning logical definition from relations. *Machine Learning*, 5:239–266, 1990.
26. M. Sebag. A constraint-based induction algorithm in FOL. In W. Cohen and H. Hirsh, editors, *Proceedings of ICML-94, International Conference on Machine Learning*. Morgan Kaufmann, J1994.
27. M. Sebag. Using constraints to building version spaces. In L. De Raedt and F. Bergadano, editors, *Proceedings of ECML-94, European Conference on Machine Learning*. Springer Verlag, 1994.
28. M. Sebag. Delaying the choice of bias: A disjunctive version space approach. In L. Saitta, editor, *Proceedings of the 13^{th} International Conference on Machine Learning*, pages 444–452. Morgan Kaufmann, 1996.

29. M. Sebag and C. Rouveirol. Induction of maximally general clauses compatible with integrity constraints. In S. Wrobel, editor, *Proceedings of ILP-94, International Workshop on Inductive Logic Programming*, 1994.
30. M. Sebag and C. Rouveirol. Constraint inductive logic programming. In L. de Raedt, editor, *Advances in ILP*, pages 277–294. IOS Press, 1996.
31. M. Sebag, C. Rouveirol, and J.F. Puget. ILP + stochastic bias = polynomial approximate learning. In *Proceedings of* 3^{rd} *International Workshop on MultiStrategy Learning*. MIT Press, 1996, to appear.
32. B.D. Smith and P.S. Rosembloom. Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version space. In *Proceedings of AAAI-90*, pages 848–853. Morgan Kaufmann, 1990.
33. A. Srinivasan and S. Muggleton. Comparing the use of background knowledge by two ILP systems. In L. de Raedt, editor, *Proceedings of ILP-95*. Katholieke Universiteit Leuven, 1995.
34. A. Srinivasan, personal communication.
35. L.G. Valiant. A theory of the learnable. *Communication of the ACM*, 27:1134–1142, 1984.
36. M.L. Wong and K.S. Leung. Combining genetic programming and inductive logic programming using logic grammars. In D. B. Fogel, editor, *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pages 733–736. IEEE Press, 1995.
37. J.-D. Zucker and J.-G. Ganascia. Selective reformulation of examples in concept learning. In W. Cohen and H. Hirsh, editors, *Proc. of* 11^{th} *International Conference on Machine Learning*, pages 352–360. Morgan Kaufmann, 1994.

This article was processed using the LaTeX macro package with LLNCS style