Macro-operators Revisited in Inductive Logic Programming

Érick Alphonse

MIG - INRA/UR1077
78352 Jouy en Josas CEDEX
FRANCE
ealphons@jouy.inra.fr

Abstract. For the last ten years a lot of work has been devoted to propositionalization techniques in relational learning. These techniques change the representation of relational problems to attribute-value problems in order to use well-known learning algorithms to solve them. Propositionalization approaches have been successively applied to various problems but are still considered as ad hoc techniques. In this paper, we study these techniques in the larger context of macro-operators as techniques to improve the heuristic search. The macro-operator paradigm enables us to propose a unified view of propositionalization and to discuss its current limitations. We show that a whole new class of approaches can be developed in relational learning which extends the idea of changes of representation to more suited learning languages. As a first step, we propose different languages that provide a better compromise than current propositionalization techniques between the building cost of macro-operators and the learning cost. It is known that ILP problems can be reformulated either into attribute-value or multi-instance problems. With the macro-operator approach, we see that we can target a new representation language we name multi-table. This new language is more expressive than attribute-value but less expressive than multi-instance. Moreover, it is PAC-learnable under weak constraints. Finally, we suggest that relational learning can benefit from both the problem solving and the attribute-value learning community by focusing on the design of effective macro-operator approaches.

1 Introduction

After [1], concept learning is defined as search : given a hypothesis space defined a priori, identified by its representation language, find a hypothesis consistent with the learning data. This paper, relating concept learning to search in a space state, has enabled machine learning to integrate techniques from problem solving, operational research and combinatorics. The search is NP-complete for a large variety of languages of interest (e.g. [2–4]) and heuristic search is crucial for efficiency. If heuristic search has been showed effective in attribute-value languages, it appeared early that learning in relational languages, known for more

than a decade as Inductive Logic Programming (ILP), had to face important *plateau phenomenas* [5–7] : the evaluation function, used to prioritize nodes in the refinement graph is constant in parts of the search space, and the search goes blind. These plateau phenomenas are the pathological case of heuristic search.

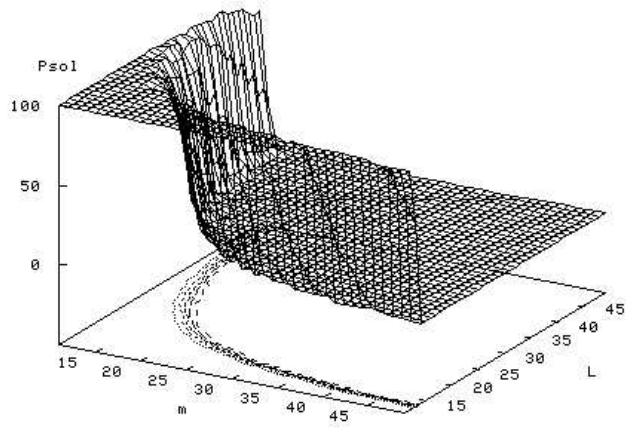

Fig. 1. (from [8]) Coverage probability P_{sol} of an example with L constants by a hypothesis with m literals (built from 10 variables). The contour level plots, projected onto the plane (m, L), correspond to the region where P_{sol} is between 0.99 and 0.01

An explanation can be given after the seminal work of [8], who studied the ILP coverage test within the phase transition framework. As illustrated in figure 1, the covering test is NP-complete and therefore exhibits a sharp phase transition in its coverage probability [9]. If one studies the probability of covering an example of fixed size by a hypothesis given the hypothesis size, one distinguishes three well-identified regions: a region, named "yes", where the probability of covering an example is close to 1, a region, named "no", where the probability is close to 0, and finally the phase transition where an example may or may not be covered. As the heuristic value of a hypothesis depends on the number of examples covered (positive or negative), we see that the two regions "yes" and "no" represent plateaus that need to be crossed during search without an informative heuristic value.

The state of the art on evaluation functions used in learning (see for example [10]) shows that all of them are based on two main parameters that are the rate of positive and negative examples covered. As these two parameters are inherited from the definition of the learning task, it is unlikely that a solution consists in designing new evaluations functions. This problem has been well studied in the problem solving community [11] and the solution proposed is based on macro-operators. Macro-operators (macros for short) are refinement operators defined by composition of elementary refinement operators. They are able to

apply several elementary operators at a time (adding or removing several literals in a hypothesis in the case of relational learning) and therefore are likely to cross non informative plateaus. The application of macros in relational learning is not new and has been investigated in e.g. [5–7, 12]. In these systems, macros *are added* to the list of initial refinement operators and are then classically exploited to search in the refinement graph. In general, this type of technique is known in Machine Learning as Constructive Induction [13], or Predicate Invention in ILP. However, in practice the induced growth of the hypothesis space leads to a decrease of performances [14].

As opposed to these approaches, we view macros as a mean to *simplify* the search space by considering only macros as elementary operators. In other words, our focus is only on the sub-graph generated by the macros. We propose to study a macro-operator approach with respect to the complexity of the new representation language of the hypothesis space associated with this sub-graph. Our approach is similar to abstraction by means of macro-operators [15, 11].

We are going to show that a large set of ILP approaches, named *propositionalization* after [16], implicitly use macros to select a sub-graph whose representation language is equivalent to attribute-value (AV). In doing so, they can delegate the simplified learning problems to well-known AV algorithms [17, 18, 16, 19–23]. The advantage of formalizing theses approaches in terms of macro-operators is threefold. Firstly, it allows to clarify and motivate the propositionalization approaches, which are still considered as ad hoc techniques in literature, as techniques to improve the heuristic search. Secondly, by drawing a parallel with macro-operators, it allows ILP to benefit from techniques formalized and developed in this framework by the Problem Solving community (see e.g. [25, 11]). Finally and most importantly, this formalization, by showing propositionalization as a two-stage process (creation of macros and delegation of the simplified search space to simpler learning algorithms), points to promising extensions. Propositionalization as proposed in the literature appears to be a special case of resolution by macro-operators. If targeting AV languages is interesting because of the subsequent use of efficient learning algorithms, the cost of building macro-operators for this language is prohibitive and equivalent to the cost of "classical" ILP systems[1] on very simple cases, as we will show it. We propose to relax the constraint on the hypothesis space induced by the definition of macros, by allowing more expressive languages, to offer a better trade-off between the cost of building macros and the cost of learning.

In the next section, before discribing the use of macro-operators to solve ILP problems, we are going to recall the different works of various researchers to delegate the resolution of ILP problems to simpler learning algorithms, namely AV and multi-instance ones, without loss of information [26, 27, 3, 29–32]. We will use this section to introduce the notations used in the rest of the paper. In section 3, we will illustrate the macro-operator approach and we will show how we can associate a language to the hypothesis sub-space selected by macros. A set

[1] We refer here "classical" ILP systems as learning systems not built on macro-operators.

of languages that can be chosen by means of macros will be presented. We will then show in section 5 that the language used by current propositionalization approaches, which is a determinate language, cannot efficiently delegate learning to AV algorithms. In section 6, we will discuss the k-locale language [3] that appears to be the first promising relaxation of the hypothesis space language. We will show that this language allows to reformulate the initial ILP problem into a new representation language we name *multi-table*. This language is simpler than the multi-instance language [33]. Moreover, in a more constraint form of the language, we will see that a monomial is PAC-learnable, which suggests that efficient algorithms exist. In section 7, we generalize further the hypothesis subspace language to the k-free and k-indeterminate languages and discuss shortly the properties of propositionalization in these languages. Finally, we will conclude and argue that the important effort put in propositionalization techniques can be pursued in richer languages that lift their current shortcomings.

2 Propositionalization

[26, 27, 34] showed that the ij-determinate language (determinate for short), restricting both the hypothesis space, $\mathcal{L}_h$, and the instance space, was the stronger language bias that could be applied to ILP. Indeed, they proved that it was as expressive as an AV language and that it could be compiled in polynomial time into it. This language has been used as learning language in LINUS and DINUS [35]. Their strategy is to change the representation of the determinate ILP problem to AV, then to delegate learning to AV algorithms and finally to write the learnt theory back to the determinate language. The work of Lavrač and Džeroski has been fruitful to ILP because it suggested that the AV expertise could be reused by change of representation of the initial learning problem.

Extensions of this approach have been proposed in more expressive languages which introduce indeterminism [3, 29–32]. In this case, the reformulated ILP problem is not an AV problem but a multi-instance one. We now give a general description of the change of representation. These changes of representation are often termed propositionalization in the literature, in a more specific way than [16], as it refers only to the change of representation, independently of the use of macro-operators. We will use this more specific meaning in the rest of the paper in order to better distinguish the different techniques presented in the next section.

A prerequisite to delegate the resolution of an ILP problem to AV or multi-instance algorithms is that the hypothesis spaces in both languages must be isomorphic. In the following, we represent such a bias as a hypothesis base (by analogy with a vectorial base), noted $\mathcal{B}$.

Definition 1 (hypothesis base). *A hypothesis base, noted $\mathcal{B}$, is of the form:*

$$\mathcal{B} = (tc \leftarrow < l_1, \ldots, l_n, v_1, \ldots, v_m >)$$

with $l_1, \ldots, l_n$ a set of literals and $v_1, \ldots, v_m$ a set of constraint variables in $l_1, \ldots, l_n$. Each hypothesis has the head tc and its body is represented by a vector in $\mathcal{B}$.

By construction, there is a one-to-one mapping between a hypothesis in the clausal space and a hypothesis in the AV space. To perform the search in the AV search space, the ILP instance space needs to be reformulated to emulate the covering test. In other words, the change of representation hard-codes the covering test between the hypotheses and the examples by computing all matchings between $\mathcal{B}$ and the ILP examples for subsequent use by the AV or multi-instance algorithms. If the language is determinate, there will be only one matching and therefore an example will be reformulated into one AV vector, and in the case where the language is indeterminate, an example will be reformulated into a bag of vectors. The fact that the new instance space in this latter case is a multi-instance space is due to the definition of the covering test in ILP: to cover an ILP positive example, we need to find one matching substitution, and to reject an ILP negative example, we need not find any matching substitutions.

An example of propositionalization of a toy problem a la Michalski's trains under θ-subsumption is given in table 1 with the base:

$$\mathcal{B} = (west(T) \leftarrow < car(T, C1), roof(C1), \#loads(C1, 2), car(T, C2), \#loads(C2, N), N >)$$

As the train representation is indeterminate, the new representation obtained by propositionalization is a multi-instance one. For a better reading, we omit the instantiation of the head variables of $\mathcal{B}$, as they are determinate.

Table 1. A train problem and its propositionalization given $\mathcal{B}$

Examples	Background Knowledge
e^+ $west(t1)$	$car(t1, c11).$ $roof(c11).$ $\#loads(c11, 2).$
e^- $west(t2)$	$car(t1, c12).$ $short(c12).$ $\#loads(c12, 2).$
	$car(t2, c21).$ $short(c21).$ $\#loads(c21, 4).$
	$car(t2, c22).$ $rect(c22).$ $\#loads(c22, 2).$

	Variables		Attributes					
	$C1$	$C2$	$car(T, C1)$	$roof(C1)$	$\#loads(C1, 2)$	$car(T, C2)$	$\#loads(C2, N)$	N
e^+	$c11$	$c11$	T	T	T	T	T	2
	$c11$	$c12$	T	T	T	T	T	2
	$c12$	$c11$	T	F	T	T	T	2
	$c12$	$c12$	T	F	T	T	T	2
e^-	$c21$	$c21$	T	T	F	T	T	4
	$c21$	$c22$	T	T	F	T	T	2
	$c22$	$c21$	T	F	T	T	T	4
	$c22$	$c22$	T	F	T	T	T	2

As noted in [36], the propositionalization is not tractable in the general case: the covering test being NP-complete, the reformulation of a single relational ex-

ample e can yield an exponential number of AV vectors with respect to the size of $\mathcal{B}$ and e. However, propositionalization has been used successfully in numerical learning. As we can see in the example above, $\mathcal{B}$ introduces the constraint variable N and constraints can be learnt by a multi-instance algorithm. This propositionalization dedicated to numerical learning, which can be traced back to INDUCE [37], has been developed in [29, 38–40]. In the rest of the paper, we are going to show that this is one of the advantages that motivates the definition of macro-operators in richer languages than the determinate language.

3 Solving ILP problems by means of macro-operators

As mentioned previously, macro-operators have already been used in ILP [5–7, 12]. We illustrate their approach in an example showing a plateau at the first refinement step. For lack of space, we assume the reader is familiar with ILP (see e.g. [41]).

Example 1 *We run a top-down greedy algorithm a la FOIL [5] on the following train-like problem:*

Examples	*Background Knowledge*		
e_1^+ $west(t1)$	$car\_1(t1, c11)$.	$short(c11)$.	$car\_2(t1, c12)$.
e_2^+ $west(t2)$	$roof(c12)$.	$car\_1(t2, c21)$.	$short(c21)$.
e_1^- $west(t3)$	$car\_2(t2, c22)$.	$short(c22)$.	$car\_1(t3, c31)$.
	$roof(c31)$.	$car\_2(t3, c32)$.	$roof(c32)$.

It has two positive examples and one negative example. The semantic is classical and for example the first positive example reads : the train goes west, its first car is short and its second car has a roof. The following hypothesis is consistent with the data:

$$west(T) \leftarrow car\_1(T, C1), short(C1)$$

The search starts with a hypothesis with an empty body and considers the possible refinements to specialize it. The two literals $car\_1(T, C1)$ and $car\_2(T, C2)$ alone do not discriminate between the positive and negative examples (each train having two cars) and then have the same heuristic value (a zero information gain). The learning algorithm has to make a choice and for example it may add $car\_2(T, C2)$, which does not lead to a solution, when the conjunction $car\_1(T, C1), short(C1)$ discriminates between the classes. The solution to prevent the lack of heuristic information is to define a macro-operator in order to specialize the hypothesis with this conjunction, crossing the plateau.

One solution we adopt to define the macro-operators, which is also discussed in [6], is to create a new literal whose definition corresponds to the composition of the elementary refinement operators. By adding this literal to the background knowledge by saturation (see e.g. [42]), the initial refinement operators can use this literal to refine a hypothesis. Classically, we define a macro-operator with respect to the hypothesis language, its definition corresponding to a valid sequence of refinement operators.

Definition 2 (macro-operator). *A macro-operator is a predicate m. The definition of m is a set of clauses $\{D_i\}$ having the head built on m. Each clause is built on the hypothesis language $\mathcal{L}_h$, i.e.:*

$$\forall i, \exists h \in \mathcal{L}_h \text{ s.t. } body(D_i) = body(h)$$

Example 2 *We use the prolog notation of predicate definition to define the two following macros:*

$$car\_1\_short(T) :- car\_1(T, C1), short(C1).$$
$$car\_2\_roof(T) :- car\_2(T, C1), roof(C1).$$

The new literals are added to the background knowledge if their definition holds in this background knowledge (saturation), and we have the new learning problem:

Examples	*Background Knowledge*			
e_1^+ $west(t1)$	$car\_1(t1, c11)$.	$short(c11)$.	$car\_2(t1, c12)$.	$roof(c12)$.
e_2^+ $west(t2)$	$car\_1(t2, c21)$.	$short(c21)$.	$car\_2(t2, c22)$.	$short(c22)$.
e_1^- $west(t3)$	$car\_1(t3, c31)$.	$roof(c31)$.	$car\_2(t3, c32)$.	$roof(c32)$.
	car_1_short(t1).	**car_2_roof(t1)**.	**car_1_short(t2)**.	**car_2_roof(t3)**.

Here, we propose to simplify the representation by using only the newly defined literals. In other words, we are not interested in the refinement graph augmented with the macros, but only by the sub-graph generated by the macros. If we consider again the above example with this approach, we have the new representation:

Examples	Background knowledge
e_1^+ $west(t1)$	$car\_1\_short(t1)$. $car\_2\_roof(t1)$.
e_2^+ $west(t2)$	$car\_1\_short(t2)$. $car\_2\_roof(t3)$.
e_1^- $west(t3)$	

We can notice now that only the constants of the examples are in the background knowledge. This language is remarkable because it is a determinate language and is known to be as expressive as the attribute-value language, as explained in the previous section. The search can then be delegated to an AV algorithm by propositionalizing the learning examples. This reformulation is shown in table 2.

Table 2. Reformulation of a determinate ILP problem into an AV problem

	Variables	Attributes	
	T	$car\_1\_short(T)$	$car\_2\_roof(T)$
e_1^+	$t1$	T	T
e_2^+	$t2$	T	F
e_1^-	$t3$	F	T

This example is representative of ILP problem solving by macro-operators that we propose and we are going to later give a general algorithm (section 4). As far as we know, the first works on this approach, even if they do not

refer to macros, are due to [17, 18]. A lot of subsequent work has been done [16, 19–23] with the same principle of defining macros to simplify the problem into a determinate language, then propositionalizing to delegate learning to AV algorithms. We do not discuss the different techniques that they propose to build what we refer to as macros as it is out of the scope of the paper.

Viewing these approaches as a two-stage process points out interesting directions. On the one hand, it unifies all the above-mentioned approaches focusing on the way they build the macro-operators. Moreover, it proposes a motivation for them, as they are still viewed in literature as ad hoc techniques, as a way to improve heuristic search in ILP. On the other hand, these approaches appear to be a *particular case of solving ILP problems by macros*. They define macros to target the particular determinate language, but we can develop new approaches that define macros to target more expressive languages. By doing so, we argue that we are going to yield different trade-offs between the cost of building macros and the cost of learning. For instance, we are going to show in the next section that building macros for a determinate language is as expensive as "classical" ILP on simple cases.

Even though all languages from determinate languages, actually used, to restrictions of first-order logic can be used, we restrict ourselves to relational languages where propositionalization was proposed. The restriction to these particular languages is motivated by the fact that learning algorithms for these languages are well-known and efficient and we can hope for better trade-offs. We consider all above-mentioned approaches , which have been shown competitive with respect to "classical" ILP systems, as an empirical validation.

4 The general macro-operator approach

We now give the general algorithm for solving ILP problems by means of macro-operators:

1. Choose $\mathcal{L}_\mathcal{B}$, the language in which the hypothesis base $\mathcal{B}$ will be expressed
2. Build a set of macro-operators with respect to $\mathcal{L}_\mathcal{B}$ to define $\mathcal{B}$ in $\mathcal{L}_\mathcal{B}$.
3. Propositionalize the learning data
4. Apply a learning algorithm tailored to the representation of the new learning problem
5. Write the learnt theory back to the initial ILP language[2]

Note that we mix up the saturation and propositionalization stages (step 3) as it is done in the systems proposed in the literature; the cost of stauration, which is NP-complete in the general case, is added to the cost of propositionalization. The size of the reformulated problem can be straightforwardly deduced from the hypothesis base. The size is exponential in the number of variables in the base, the variables appearing in the head excepted.

[2] This is not always possible depending on the learning algorithm, like neural networks for example. Classification of future instances must be done in the propositionalized space.

The gain that more expressive languages than determinate languages can provide to the macro-operator approach is compensated by the cost of reformulation of an ILP problem by propositionalization. Indeed, propositionalization of indeterminate languages yields multi-instance problems of exponential size as the covering test is NP-complete in these languages (section 2). A control of the cost of propositionalization must to be done by using languages with bounded indeterminism. Some of these languages have been well studied by [3] and we propose to use them as a basis for new languages for the macro-operator approach. The resulting expected trade-offs between the cost of building macros and the cost of learning is presented in figure 2. The cost of building macro-operators has to be understood as the complexity of defining macros in order to obtain a consistent representation of the new problem.

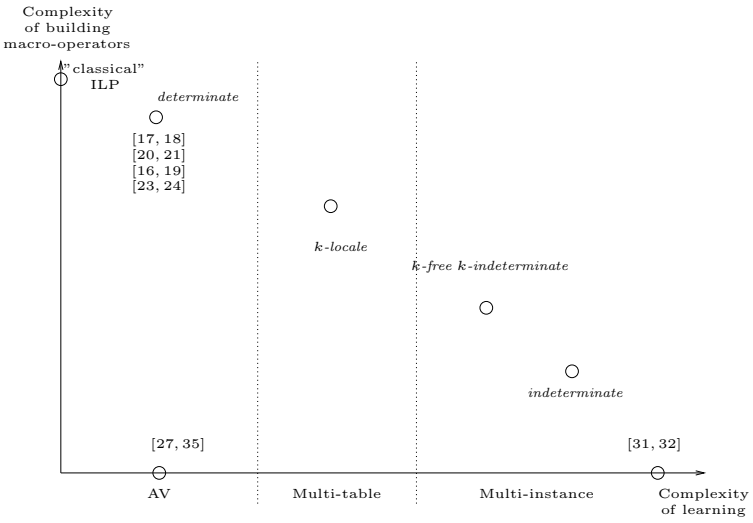

Fig. 2. Trade-offs between cost of building macros and cost of learning

On one side of the range of possible trade-offs, we have "classical" ILP systems. Indeed, we can see a clausal theory as a set of macro-operators[3] and a propositionalization can be performed. Learning is straightforward, classifying a new example as positive if one of its attributes is true, and negative otherwise, that is to say if at least one of the clauses covers the example[4]. This exemple illustrate the cost of building the macros necessary to obtain a trivial representation of the examples.

[3] This approach has been used early in ILP and has been refined recently in the "repeat learning framework" [14].

[4] This is very similar to the table of macro-operators defined by [11] who notes that once the table is computed, no search has to be done to solve the problem.

On the opposite side, we have the approaches of [31, 32] that do not build any macro, or more precisely in our framework, define directly a macro as a literal. The cost of learning is then entirely delegated to a multi-instance algorithm which has to deal with data of exponential size in the general case.

In between these two extreme sides, we see the different classes of approaches which depend on the expressivity of the representation language of the hypothesis base. In the next section, we discuss the limitations of the first class that targets the determinate language to delegate some cost to AV algorithms [17, 18, 16, 19–23]. We then evaluate the interest of the languages we propose, focusing on the k-locale language which is the first promising relaxation of the determinate language.

5 Limitations of the determinate language

Intuitively, the cost of building macros in order to get a consistent representation of the learning problem in a determinate language is very high. It comes from the fact that the new representation describe the example as a whole, as a unique entity, what [20] named an individual-centered representation. That is to say, the macros must define global properties of the learning example. If we have a look at a macro for the determinate language :

$$short\_car\_w/roof(T) : -car(T, C1), short(C1), roof(C1)$$

We see that it describes a property of the train, of the example as a whole: the train has the property of having a short car with a roof. This impacts the cost on two ways.

5.1 Complexity of numerical learning

The numerical learning capacity is very limited. We can have a macro describing the total number of loads that the train carries, but not the number of loads carried by a car of the train for example. In other words, the numerical learning part delegated to the learning algorithm is limited only to the properties of the example as a whole. For example, as a workaround, the RELAGGS system [21] computes statistics like the average and the maximum of numerical values appearing in the example. To do numerical learning on parts of the example, the ILP system must generate macro-operators such as:

$$car\_2 + \_loads(T) : -car(T, C1), \#loads(C1, N), N \geq 3$$

to learn that a car must carry more than 2 loads.

A solution to this problem is to delegate the complexity of numerical learning to multi-instance algorithms as in [29, 30] for example. Numerical variables are then introduced in the hypothesis base (introducing indeterminism) and the set of all their matchings is gathered under a tabular format. As an example,

learning constraints on the number of car loads is allowed by the definition of the macro:

$$\#car\_loads(T, N) : -car(T, C1), \#loads(C1, N)$$

The associated hypothesis base $\mathcal{B} = (west(T) \leftarrow < \#car\_loads(T, N), N >)$ would give the following reformulation on a hypothetic ILP problem:

	Variables	Attributes	
	T	$\#car\_loads(T, N)$	N
e^+	$t1$	T	1
	$t1$	T	3
	$t1$	T	2
e^-	$t2$	T	1
	$t2$	T	2
	$t2$	T	2

A multi-instance algorithm could then induce the concept definition:

$$west(T) \leftarrow \#car\_loads(T, N), N \geq 3$$

5.2 Size of macro-operators

A second limitation is that some concepts cannot be learnt by AV algorithms and therefore must be represented by a single macro. The cost of building the macros is then *as expensive as* the cost of a classical ILP algorithm. We show this case in a simple example. Let us assume that we have to learn the following concept:

$$west(T) \leftarrow car(T, C1), rect(C1), car(T, C2), rect(C2), C1 \neq C2$$

The concept is that a train has two different rectangular cars. We can see that it cannot be learned by building macros defined with a subset of the concept's literals. The only way to prevent the instantiations of the two cars on a same car is to form a unique macro equivalent to the definition of the concept:

$$2\_rect\_cars(T) : -car(T, C1), rect(C1), car(T, C2), rect(C1), C1 \neq C2$$

By relaxing the constraint of determinism, one can build two macros less complex, defining a rectangular car each, and obtain a multi-intance problem.

We see that in both cases the use of a richer representation language allows to better use the learning algorithms. The first promising language that we consider is the k-locale language.

6 The *k*-locale language

The k-locale language has been proposed by [3] to introduce a bounded determinism in each locale of a clause. A locale is defined as a maximal set of literals where their free variables do not appear in other locales. By bounding

the maximum number of literals in a locale by k, the covering test complexity of a k-locale clause is exponential in k, the covering test of each locale being independant. In this language, the hypothesis base is a k-locale of the form $\mathcal{B} = (cc \leftarrow < LOC_1, \ldots, LOC_n >)$, where each LOC_i, a conjounction of macros, is a locale. However, as opposed to the definition of [3], the number of matching substitutions of a locale depends here on its number of variables (the saturation process being implicit). We therefore redefine the notion of locality as follows:

Definition 3 (k-locale base). *Let the k-locale hypothesis base be:*

$$\mathcal{B} = (tc \leftarrow < LOC_1, \ldots, LOC_n >)$$

We have $\forall i \in \{1, \ldots, n\}, LOC_i$ is a locale and $k \geq (vars(LOC_i) \backslash vars(tc))$

This language allows to define macros that introduce a bounded indeterminism at the level of each locale of $\mathcal{B}$. For example, $\mathcal{B}$ is a 2-locale with its 2-locales underlined:

$$\mathcal{B} = (west(T) \leftarrow \quad < \underline{short\_car(T, C1), roof(C1),} \\ \underline{short\_car(T, C2), \#loads(C2, N), N} >)$$

This definition of locality is similar to [43], who used the principal of locality as a decomposition technique to improve the covering test. This decomposition technique is classical to decompose a problem into a set of sub-problems (see e.g. [44]), and we will see the additional benefit of k-locale bases on propositionalization.

By definition of the locality, while propositionalizing an ILP problem with $\mathcal{B}$, the instantiation of variable $C1$ does not constraint the instantiations of variable $C2$. Therefore, it is more efficient to represent the new instance space by a product representation.

Let us consider the following train problem example:

Examples	Background knowledge
e^+ $west(t1)$	$car(t1, c11)$. $roof(c11)$. $rect(c11)$.
e^- $west(t2)$	$car(t1, c12)$. $short(c12)$. $\#loads(c12, 2)$.
	$car(t2, c21)$. $short(c21)$. $roof(c21)$.
	$car(t2, c22)$. $rect(c22)$. $\#loads(c22, 3)$.

The propositionalization must take into account the locales to generate for each of them a multi-instance problem as we show in figure 3. This representation of the problem has only two lines at the maximum per example, instead of four in its developed representation, which would be obtained by naive propositionalization. This representation, obtained by decomposition of a multi-instance problem, is simpler and exponentially more compact than the developed multi-instance problem. We name this new representation the *multi-table* representation, each table being a multi-instance problem. Note that the multi-table representation is not a generalization of the multi-instance representation, but is between AV and multi-instance, as a result of a product representation of a multi-instance problem, like other decomposition techniques (see [43, 44]). We present some learnability result in this language.

	Variables	Attributes	
	$C1$	$sh\_c(T, C1)$	$roof(C1)$
e^+	$c11$	F	T
	$c12$	T	F
e^-	$c21$	T	T
	$c22$	F	F

$\times$

	Variables	Attributes		
	$C2$	$sh\_c(T, C2)$	$\#loads(C2, N)$	N
e^+	$c11$	F	F	-
	$c12$	T	T	2
e^-	$c21$	T	F	-
	$c22$	F	T	3

	Variables		Attributes				
	$C1$	$C2$	$short\_car(T, C1)$	$roof(C1)$	$short\_car(T, C2)$	$\#loads(C2, N)$	N
e^+	$c11$	$c11$	F	T	F	F	-
	$c11$	$c12$	F	T	T	T	2
	$c12$	$c11$	T	F	F	F	-
	$c12$	$c12$	T	F	T	T	2
e^-	$c21$	$c21$	T	T	T	F	-
	$c21$	$c22$	T	T	F	T	3
	$c22$	$c21$	F	F	T	F	-
	$c22$	$c22$	F	F	F	T	3

Fig. 3. A multi-table representation and its developed multi-instance one

6.1 On learnability of multi-table problems

A multi-table representation is a vector $< T_1, \ldots, T_n >$ where each coordinate is a multi-instance problem. A example is of the form $e =< b_1, \ldots, b_n >$ with b_i a bag of AV instances defined in the instance space of T_i. The number of instances in each bag is variable and depends on the table.

No learning algorithms are devoted to this representation yet, but it is easy to extrapolate with the expertise gained from the design of multi-instance algorithms from AV algorithms. As with the multi-instance representation, it is easy to adapt top-down generate-and-test algorithms as noted in [45, 46] : these algorithms search the same hypothesis space but adapt the covering test to the multi-table representation.

Much in line with Cohen's work on PAC-learnability of single clause in the k-locale language, an interesting restriction of the multi-table representation is to bound the number of attributes by a constant l, for any number of tables. In the macro-operator approach, this restriction corresponds to bounding the size of the locales in the hypothesis base, but not its size. In this language, we show that monomials are PAC-learnable and then efficient algorithms exist. We give the proof for completeness as we work with the multi-table representation and that we do not have the same parameters as Cohen (l and k). Following, [3], this concept language is noted $\mathcal{L}^1_{l-MULTI-TABLE}$.

Theorem 1 *For any fixed l, the language family $\mathcal{L}^1_{l-MULTI-TABLE}$ is PAC-learnable.*

Proof 1 *We use the proof technique given by [47] (lemma 1): we reduce learning in $\mathcal{L}^1_{l-MULTI-TABLE}$ to learning monomials in a boolean space, which is known to be PAC-learnable.*

Lemma 1 (from [47]) *Let $\mathcal{L}_{h1}$ and $\mathcal{L}_{h2}$ be two hypothesis languages, $\mathcal{L}_{e1}$ and $\mathcal{L}_{e2}$ their associated languages of examples and $C \in \mathcal{L}_{h1}$ the target concept. Let $f_i : \mathcal{L}_{e1} \rightarrow \mathcal{L}_{e2}$ et $f_c : \mathcal{L}_{h1} \rightarrow \mathcal{L}_{h2}$ be the two reduction functions.*

If $\mathcal{L}_{h1}$ reduces to $\mathcal{L}_{h2}$ and $\mathcal{L}_{h2}$ is PAC-learnable and ${f_c}^{-1}(C)$ is computable in polynomial time then $\mathcal{L}_{h1}$ is PAC-learnable.

We consider that all tables have l attributes, possibly completed with neutral attributes for learning. We build a boolean representation of the multi-table problem by associating to each hypothesis of the search space a boolean attribute. As the hypothesis space is a direct product of the hypothesis spaces of each table, the boolean representation has $n \times 2^{2l}$ attributs. To obtain those attributes, for each table T_i containing the l boolean attributes $\{t_{i1}, \ldots, t_{il}\}$, we define a set of $2l$ boolean attributes $A_i = \{t_{i1}, \ldots, t_{il}, \neg t_{i1}, \ldots, \neg t_{il}\}$. For each set A_i, we define a set B_i of 2^{2l} attributes, each attribute corresponding to a hypothesis of T_i built on A_i.

f_i : *Each example is redescribed with the $n \times 2^{2l}$ attributes $(B_1, \ldots, B_n)$ where an attribute is true if the corresponding hypothesis covers the example and false otherwise.*

f_c : *A concept is of the form $< c_1, \ldots, c_n >$. By construction, each c_i matches an attribute $b \in B_i$ and c_i covers the example iff b is true. f_c preserve the concept extension and is easily reversible in polynomial time.*

7 k-free, l-indeterminate languages

We consider two other extensions, adapted from [3], which also bound the indeterminism of the covering test, therefore yielding bounded multi-instance problems through propositionalization. These are the k-free and the l-indeterminate languages, and they are complementary in the way they bound the indeterminism. The first language, bounding the number of free variables in $\mathcal{B}$, bounds theoretically the size of the multi-instance problem by an exponential in k. The second one directly bounds the size of the multi-instance problem by l, which is the number of matchings between $\mathcal{B}$ and the examples. As opposed to the former, it takes into account the background knowledge and the type of the variables, and therefore allows for a tighter upper bound.

As an example, one simple and efficient use of these languages is to introduce indeterminism only with the numerical variables to delegate numerical learning to multi-instance algorithms.

8 Conclusion

It is known that ILP is prone to plateau phenomenas during heuristic search and macro-operators are well motivated to reduce these problems. We have proposed a paradigm to solve ILP problems by means of macro-operators. As opposed to

previous approaches, the macros are not used to extend the refinement graph but only to select a relevant sub-graph. We have shown that we can associate a representation language to the sub-graph and study a macro-operator approach in terms of the expressivity of this representation language. We have shown that a large set of ILP techniques [17, 18, 16, 19–23] are a particular case of the paradigm. They define macros to reformulate ILP problems in a determinate language and delegate learning by propositionalization to AV learning algorithms, suitable for determinate languages. However, their cost of building macros is in the worst case as expensive as "classical" ILP. Notably, numerical learning has to be done in the ILP search space and cannot be delegated to AV algorithms. We have then proposed a range of different languages adapted from [3] that allows better trade-offs between the cost of building macros and the cost of learning.

The first promising trade-off is the k-locale language that reformulates ILP problems to a new representation language we have named *multi-table*. It is a product representation of a multi-instance problem and can represent the same amount of information in a more compact way. Moreover, we have shown that monomials was PAC-learnable if the number of attributes per table was fixed, which suggests that efficient learning algorithms are available. Such a restriction appears naturally in the propositionalization framework if we limit the size of each locale in the hypothesis space. We think that the definition of the multi-table representation goes beyond its use in ILP and may be used as learning language as well.

From our point of view, the important effort put in designing macro-operators for the determinate language for the last ten years can be pursued in richer languages that lift their current shortcomings. ILP can benefit from the problem solving community by reusing techniques to build macro-operators and then delegate learning to simpler learning algorithms like AV, multi-table and multi-instance algorithms. To start experimenting with the different classes of approach, we work on a multi-table algorithm and we plan to develop techniques to build macro-operators for the k-locale language.

References

1. Mitchell, T.M.: Generalization as search. Artificial Intelligence **18** (1982) 203–226
2. Haussler, D.: Learning conjunctive concepts in structural domains. Machine Learning **4** (1989) 7–40
3. Cohen, W.W.: Learnability of restricted logic programs. In Muggleton, S., ed.: Proceedings of the 3rd International Workshop on Inductive Logic Programming, J. Stefan Institute (1993) 41–72
4. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. The MIT Press, Cambridge, Massachusetts (1994)
5. Quinlan, J.R.: Determining literals in inductive logic programming. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence, Sydney, Austalia (1991) 746–750
6. Silverstein, G., Pazzani, M.J.: Relational cliches: Constraining constructive induction during relational learning. In Birnbaum, L., Collins, G., eds.: Proceedings of

the 8th International Workshop on Machine Learning, Morgan Kaufmann (1991) 203–207
7. Richards, B., Mooney, R.: Learning relations by pathfinding. In: Proceedings of the Tenth National Conference on Artificial Intelligence. (1992) 723–738
8. Giordana, A., Saitta, L.: Phase transitions in learning relations. Machine Learning **41** (2000) 217–25
9. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In Myopoulos, John; Reiter, R., ed.: Proceedings of the 12th International Joint Conference on Artificial Intelligence, Sydney, Australia, Morgan Kaufmann (1991) 331–340
10. Fürnkranz, J., Flach, P.: An analysis of rule learning heuristics. Technical Report CSTR-03-002, Department of Computer Science, University of Bristol (2003)
11. Korf, R.E.: Macro-operators: a weak method for learning. Artificial Intelligence, 1985 **26** (1985) 35–77
12. Peña Castillo, L., Wrobel, S.: Macro-operators in multirelational learning: a search-space reduction technique. In: Proceedings of the 13th European Conference on Machine Learning. Volume 2430 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2002) 357–368
13. Michalski, R.S.: Pattern recognition as knowledge-guided computer induction. Technical Report 927, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois (1978)
14. Khan, K., Muggleton, S., Parson, R.: Repeat learning using predicate invention. In Page, C., ed.: Proc. of the 8th International Workshop on Inductive Logic Programming (ILP-98). LNAI 1446, Berlin, Springer-Verlag (1998) 165–174
15. Amarel, S. In: On Representations of Problems of Reasoning about Actions. American Elsevier, New York, NY (1968) 131–171
16. Kramer, S., Pfahringer, B., Helma, C.: Stochastic propositionalization of non-determinate background knowledge. In Page, D., ed.: Proc. of the 8th International Workshop on Inductive Logic Programming, Springer Verlag (1998) 80–94
17. Turney, P.: Low size-complexity inductive logic programming: The East-West challenge considered as a problem in cost-sensitive classification. In: Proceedings of the 5th International Workshop on Inductive Logic Programming. (1995) 247–263
18. Geibel, P., Wysotzki, F.: Learning context dependent concepts. In Raedt, L.D., ed.: Proceedings of the 5th International Workshop on Inductive Logic Programming, Department of Computer Science, Katholieke Universiteit Leuven (1995) 323–337
19. Cumby, C., Roth, D.: Relational representations that facilitate learning. In Cohn, A.G., Giunchiglia, F., Selman, B., eds.: Proceedings of the Conference on Principles of Knowledge Representation and Reasoning (KR-00), S.F., Morgan Kaufman Publishers (2000) 425–434
20. Lavrač, N., Flach, P.A.: An extended transformation approach to inductive logic programming. ACM Transactions on Computational Logic **2** (2001) 458–494
21. Krogel, M.A., Wrobel, S.: Transformation-based learning using multirelational aggregation. In: Proceedings of the 11th International Conference on Inductive Logic Programming. Volume 2157 of LNAI., Springer-Verlag (2001) 142–155
22. Roth, D., Yih, W.: Relational learning via propositional algorithms: An information extraction case study. In Nebel, B., ed.: Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01), San Francisco, CA, Morgan Kaufmann Publishers, Inc. (2001) 1257–1263

23. Kramer, S., de Raedt, L.: Feature construction with version spaces for biochemical applications. In: Proc. 18th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA (2001) 258–265
24. Liquière, M.: Du structurel au propositionnel, une approche formelle. In: CAP, Grenoble (2001)
25. Hernádvölgyi, I.T.: Searching for macro operators with automatically generated heuristics. Lecture Notes in Computer Science **2056** (2001) 194–??
26. Lavrac, N., Dzeroski, S., Grobelnik, M.: Learning nonrecursive definitions of relations with LINUS. In Kodratoff, Y., ed.: Proceedings of the European Working Session on Learning : Machine Learning (EWSL-91). Volume 482 of LNAI., Porto, Portugal, Springer Verlag (1991) 265–281
27. Dzeroski, S., Muggleton, S., Russell, S.J.: PAC-learnability of determinate logic programs. In: Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT-92), Pittsburgh, Pennsylvania, ACM Press (1992)
28. Cohen, W.W.: Rapid prototyping of ILP systems using explicit bias. In Bergadano, F., Raedt, L.D., Matwin, S., Muggleton, S., eds.: Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming, Morgan Kaufmann (1993) 24–35
29. Zucker, J.D., Ganascia, J.G.: Selective reformulation of examples in concept learning. In: Proc. 11th International Conference on Machine Learning, Morgan Kaufmann (1994) 352–360
30. Sebag, M., Rouveirol, C.: Induction of maximally general clauses consistent with integrity constraints. In Wrobel, S., ed.: Proceedings of the 4th International Workshop on Inductive Logic Programming. Volume 237 of GMD-Studien., Gesellschaft für Mathematik und Datenverarbeitung MBH (1994) 195–216
31. Fensel, D., Zickwolff, M., Wiese, M.: Are substitutions the better examples? Learning complete sets of clauses with Frog. In: Proceedings of the 5th International Workshop on Inductive Logic Programming. (1995) 453–474
32. Sebag, M., Rouveirol, C.: Resource-bounded relational reasoning: Induction and deduction through stochastic matching. Machine Learning **38** (2000) 41–62
33. Dietterich, T.G., Lathrop, R.H., Lozano-Pérez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artificial Intelligence **89** (1997) 31–71
34. Kietz, J.U., Dzeroski, S.: Inductive logic programming and learnability. SIGART Bulletin **5** (1994) 22–32
35. Lavrač, N., Džeroski, S.: Inductive Logic Programming : techniques and Applications. Ellis Horwood (1994)
36. Sebag, M., Rouveirol, C.: Tractable induction and classification in first order logic via stochastic matching. In: 15th Int. Join Conf. on Artificial Intelligence (IJCAI'97), Morgan Kaufmann (1997) 888–893
37. Dietterich, T.G., Michalski, R.S.: A comparative review of selected methods for learning from examples. In Michalski, R.S., Carbonell, J.G., Mitchell, T.M., eds.: Machine Learning, an Artificial Intelligence approach. Volume 1. Morgan Kaufmann, San Mateo, California (1983) 41–81
38. Sebag, M., Rouveirol, C.: Constraint inductive logic programming. In Raedt, L.D., ed.: Proceedings of the 5th International Workshop on Inductive Logic Programming, Department of Computer Science, Katholieke Universiteit Leuven (1995) 181–198
39. Anthony, S., Frisch, A.M.: Generating numerical literals during refinement. In Lavrač, N., Džeroski, S., eds.: Proceedings of the 7th International Workshop on Inductive Logic Programming. Volume 1297 of LNAI., Berlin, Springer (1997) 61–76

40. Botta, M., Piola, R.: Refining numerical constants in first order logic theories. Machine Learning **38** (2000) 109–131
41. Nienhuys-Cheng, S.H., de Wolf, R.: Foundations of Inductive Logic Programming. Volume 1228 of Lecture Notes in Artificial Intelligence. Springer-Verlag (1997)
42. Rouveirol, C.: Flattening and saturation: Two representation changes for generalization. Machine Learning **14** (1994) 219–232
43. Kietz, J.U., Lübbe, M.: An efficient subsumption algorithm for inductive logic programming. In Cohen, W.W., Hirsh, H., eds.: Proc. 11th International Conference on Machine Learning, Morgan Kaufmann (1994) 130–138
44. Gottlob, G., Leone, N., Scarello, F.: A comparison of structural CSP decomposition methods. Artificial Intelligence **124** (2000) 243–282
45. de Raedt, L.: Attribute-value learning versus inductive logic programming : The missing link. In Page, D., ed.: Proc. of the 8th International Workshop on Inductive Logic Programming, Springer Verlag (1998) 1–8 Extended abstract.
46. Zucker, J.D., Chevaleyre, Y.: Comprendre et rsoudre les problmes d'apprentissage multi-instances et multi-parties. In: Journes Francophone d'Apprentissage. (1998)
47. Pitt, L., Warmuth, M.K.: Prediction preserving reducibility. J. of Comput. Syst. Sci. **41** (1990) 430–467 Special issue of the for the *Third Annual Conference of Structure in Complexity Theory* (Washington, DC., June 88).