

Multiple Predicate Learning*

Luc De Raedt¹, Nada Lavrač² and Sašo Džeroski²

¹Department of Computing Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium

² Computer Science Department, Jožef Stefan Institute
Jamova 39, 61111 Ljubljana, Slovenia

Abstract

We study multiple predicate learning in an empirical setting. Problems with existing inductive logic programming approaches in this setting are sketched and an empirical ILP system, named MPL, that avoids these problems is proposed. Preliminary experiments indicate that MPL can solve problems beyond the scope of other ILP systems.

1 Introduction

Recently, there has been an increased interest in ILP [Muggleton, 1992; De Raedt, 1992], resulting in a number of inductive systems employing first order logic as the representation formalism, e.g. MIS [Shapiro, 1983], FOIL [Quinlan, 1990], CIGOL [Muggleton and Buntine, 1988], CLINT [De Raedt, 1992], GOLEM [Muggleton and Feng, 1990], LINUS [Lavrač *et al.*, 1991], FORTE [Richards and Mooney, 1991], AUDREY [Wogulis, 1991], and MOBAL [Kietz and Wrobel, 1992]. The problem addressed by all of these systems can be formulated as follows (cf. [Muggleton, 1992]): given is a background theory B containing definitions of predicates in normal or definite clause logic and a set of positive and negative examples (true and false ground facts) $E = P \cup N$; the aim is to find a hypothesis H such that $\forall p \in P : B \cup H \models p$ and $\forall n \in N : B \cup H \not\models n$, i.e. the background theory together with the hypothesis entails all positive and none of the negative examples.

In this paper, we investigate the complications that arise when the example set contains examples for multiple predicates. We study the problem in the empirical setting, i.e. under the following assumptions: the whole example set is given initially, no oracle can be consulted, and the initial theory contains no definitions for the predicates to be learned.

The ILP literature has largely ignored this problem. Indeed, the empirical systems (such as FOIL, GOLEM,

LINUS) are typically designed for single predicate learning. To learn multiple predicates, a knowledgeable teacher has to supply different learning sets such that all predicates can be learned, which is – as we will show – a non-trivial task. The extensional systems GOLEM and FOIL, however, can solve some multiple predicate learning tasks. For other tasks they may encounter a number of difficulties as shown in Section 3.

The incremental systems (MOBAL, CIGOL, CLINT, MIS) can often cope with multiple predicates but focus the learning procedure on the most recently received example (MOBAL, CLINT, MIS) or generate questions to the user to reduce the search space (MIS, CIGOL, CLINT). These settings simplify the problem to a large degree. In the first case, the order of learning the predicates (and treating the examples) is given, which is – as we will show – one of the main problems. In the second case, the oracle reduces (or even eliminates) the chance of learning incorrect (or overly general) clauses, which is – as we will show – the other main problem.

Finally, theory revisors such as FORTE and AUDREY are able to learn multiple predicates. However, AUDREY and FORTE leave a number of unsolved problems: the result of learning depends on the order of processing the examples, AUDREY does not handle recursive predicates, and both systems prefer an initial approximate domain theory.

Despite this state of the art, one could believe that learning multiple predicates is easy and that only slight changes to systems such as FOIL and GOLEM are required to learn multiple predicates. We show that this belief is essentially false. We also outline an alternative approach, the MPL algorithm, based on FOIL. It effectively learns some predicate definitions beyond the scope of the other empirical ILP systems.

In Section 2, we formalize the problem. In Section 3, we discuss the difficulties with existing approaches when learning multiple predicates. In Section 4, we outline the MPL algorithm; in Section 5, we report on some preliminary experiments and finally, in Section 6, we conclude.

2 Learning multiple predicates

As in the usual ILP setting, cf. [Muggleton, 1992; De Raedt, 1992], a *theory* is a set of normal or definite clauses (cf. [Lloyd, 1987]), *examples* are ground facts,

*This research is part of the ESPRIT III project no. 6020 on Inductive Logic Programming. Luc is funded by the Belgian National Fund for Scientific Research, Nada and Sašo are funded by the Slovenian Ministry of Science and Technology, and Nada's stay in Leuven was funded by the Belgian State-Policy-Office Grant AI/03/05.

positive examples are true in the intended interpretation and negative examples are false. The example set E is the union of positive and negative examples $E = P \cup N$. As we are learning multiple predicates, E contains examples for m target predicates $p_1, \dots, p_m$. We also divide E, P and N into m subsets E_{p_i}, P_{p_i} and N_{p_i} according to these predicates. The predicates whose definitions are given initially in the background theory B are denoted as $q_1, \dots, q_l$ ($p_i \neq q_j$).

Definition 1 A hypothesis H_p is a set of normal clauses for a predicate p .

Definition 2 An m -hypothesis H is a set of normal clauses for m different predicates $p_1, \dots, p_m$. An m -hypothesis H is thus the union of m hypotheses H_{p_i} for $i = 1, \dots, m$, $H = \bigcup_{i=1}^m H_{p_i}$.

Definition 3 An m -hypothesis H covers an example e given a theory T iff $T \cup H \models e$.

Definition 4 Given theory T , m -hypothesis H and example set E , $\text{covers}(T, H, E) = \{e \in E \mid T \cup H \models e\}$.

Definition 5 m -hypothesis H is (globally) complete given examples E and theory T iff $\text{covers}(T, H, P) = P$.

Definition 6 m -hypothesis H is (globally) consistent given examples E and theory T iff $\text{covers}(T, H, N) = \emptyset$.

Assuming the hypothesis language of normal clauses, multiple predicate learning is defined as follows:

- **Given:**
 - a set of training examples E for m different predicates $p_1, \dots, p_m$, and
 - background theory B , containing predicate definitions for $q_1, \dots, q_l$
- **Find:** an m -hypothesis H which is globally complete and consistent with regard to E and B .

The multiple predicate learning task is denoted by a triple $\text{mpl}(B, H, E)$. It is illustrated in Example 1. A single predicate learning task is a special case of the above problem specification where $m = 1$. For a predicate p_i , this task is denoted by $\text{spl}(B, H_{p_i}, E_{p_i})$.

Notice that learning multiple predicates from examples is an important problem. One significant application is in knowledge discovery, where one faces different phenomena and needs to relate them to each other. Another interesting application is logic program synthesis.

Example 1 We illustrate the problem of multiple predicate learning on a relatively simple task where the background theory B is extensional and contains a complete set of facts for the background predicates `parent`, `female` and `male`. Completeness means that all positive and negative examples of the predicates are available to the system (negative examples are generated under the CWA). The aim is to learn `father`, `mother` and `ancestor` from the complete set of examples for these predicates, under the usual interpretation of `father`, `mother` and `ancestor`. A complete and consistent 3-hypothesis H is:

```

ancestor(X,Y) ← parent(X,Y)
ancestor(X,Y) ← parent(X,Z) ∧ ancestor(Z,Y)
father(X,Y)  ← parent(X,Y) ∧ male(X)
mother(X,Y)  ← parent(X,Y) ∧ female(X)

```

The background theory B contains:

male(prudent) ←	female(laura) ←
male(willem) ←	female(esther) ←
male(etienne) ←	female(rose) ←
male(leon) ←	female(alice) ←
male(rene) ←	female(yvonne) ←
male(bart) ←	female(katleen) ←
male(luc) ←	female(lieve) ←
male(pieter) ←	female(soetkin) ←
male(stijn) ←	female(an) ←
	female(lucy) ←

parent(bart,stijn) ←	parent(katleen,stijn) ←
parent(bart,pieter) ←	parent(katleen,pieter) ←
parent(luc,soetkin) ←	parent(lieve,soetkin) ←
parent(willem,lieve) ←	parent(esther,lieve) ←
parent(willem,katleen) ←	parent(esther,katleen) ←
parent(rene,willem) ←	parent(yvonne,willem) ←
parent(rene,lucy) ←	parent(yvonne,lucy) ←
parent(leon,rose) ←	parent(alice,rose) ←
parent(etienne,luc) ←	parent(rose,luc) ←
parent(etienne,an) ←	parent(rose,an) ←
parent(prudent,esther) ←	parent(laura,esther) ←

To elaborate our ideas in more detail, some additional terminology needs to be established. We make a distinction between *local* and *global* properties. Global properties of hypotheses and clauses are defined in the context of the entire example set E and the m -hypothesis H , whereas local properties are defined using E_{p_i} and H_{p_i} :

Definition 7 Hypothesis H_{p_i} is locally consistent iff $\text{covers}(T, H_{p_i}, N_{p_i}) = \emptyset$; and locally complete iff $\text{covers}(T, H_{p_i}, P_{p_i}) = P_{p_i}$.

Definition 8 Clause $c \in H_{p_i}$ is globally incorrect iff $\text{covers}(T, H, N) \neq \emptyset$ and $\text{covers}(T, H - \{c\}, N) \subset \text{covers}(T, H, N)$.

Definition 9 Clause $c \in H_{p_i}$ is globally irrelevant iff $\text{covers}(T, H, P) = \text{covers}(T, H - \{c\}, P)$.

A clause c is incorrect if it is responsible, either directly or indirectly, for negative examples being covered by the hypothesis H . In the special case where c is incorrect and $\text{covers}(T, H - \{c\}, N) = \emptyset$, we have that c alone is responsible for the inconsistency of the hypothesis. A clause is irrelevant if it does not contribute to positive examples being covered.

3 State-of-the-art

The application of empirical single predicate learners to a multiple predicate learning task encounters a number of problems. First, we argue that one should be cautious when applying extensional predicate learners for multiple predicate learning. Second, we show that when using a single predicate learner to learn multiple predicates, the result depends very much on the order of learning the predicates.

3.1 Extensionality

Let us first discuss the way extensional systems, such as GOLEM and FOIL, handle multiple predicate learning

problems. To solve a multiple predicate learning problem $mpl(B, H, E)$, they first transform the background theory B into a ground h -easy model M of the background theory B . Such a model contains a set of true ground facts derivable from B by a depth-bounded interpreter. To learn the predicates p_i , they then repeatedly call their inductive learning procedures on the single predicate learning tasks $spl(M \cup P, H_{p_i}, E_{p_i})$. The resulting m -hypothesis is $H = H_{p_1} \cup \dots \cup H_{p_m}$.

The approach works well provided that the example set E is complete, i.e. when E contains all possible positive and negative examples for the predicates p_i . For example, the problem specified in Example 1 can be solved by GOLEM using a complete example set. However, when the example set is incomplete, which is the usual setting in inductive learning, a number of serious problems may occur. The reason for these problems is the use of an extensional notion of coverage:

Definition 10 An m -hypothesis H extensionally covers an example e with respect to a ground model M iff there is a clause $c \in H$ and a substitution θ such that $head(c)\theta = e$ and $body(c)\theta \subset M$.

The notions of extensional completeness and consistency are derived in a straightforward manner from this definition. To distinguish the extensional notions from the ones introduced earlier, we will refer to the non-extensional ones as *intensional*.

Using extensional coverage can lead to problems because an example may be covered intensionally, but not extensionally and vice versa (see Example 2 for an illustration and [De Raedt *et al.*, 1993] for more details). Two main problems arise because of this. First, an extensional system may find a hypothesis that is extensionally globally complete and consistent, but not intensionally (see Example 2). Second, the system may be unable to find an intensionally globally complete and consistent hypothesis when there is one. Both problems disappear when care is taken that the already induced clauses are added to the theory. When working intensionally, this can easily be realized. When working extensionally, this is computationally much more expensive, as the whole model may have to be regenerated (in the case of normal program clauses).

Example 2 Reconsider the theory of Example 1, and suppose the aim is to induce the concepts `male_ancestor` (`m-a`) and `female_ancestor` (`f-a`) from father and mother and a complete example set. FOIL 2.1 induced the hypothesis below. This definition is complete and consistent from an extensional point of view (in the given database, both parents are always specified). From an intensional point of view, however, the definition is useless as it cannot be used to derive conclusions from a theory containing father and mother only.

$m-a(X, Y) \leftarrow \text{father}(X, W) \wedge \text{mother}(M, W) \wedge f-a(M, Y)$
 $f-a(X, Y) \leftarrow \text{mother}(X, W) \wedge \text{father}(F, W) \wedge m-a(F, Y)$

3.2 Order dependency

When adopting the above conclusion, the straightforward way to solve the multiple predicate learning problem $mpl(B, H, E)$ using a single predicate learner works

as follows. Multiple predicate learning starts with a 0-hypothesis, i.e. $H = \emptyset$. In each step of multiple predicate learning, the current k -hypothesis ($k < m$) $H_{cur} = H_{s_1} \cup \dots \cup H_{s_k}$ needs to be refined. At each step of target theory construction, the next predicate $s_{k+1} \in \{p_1, \dots, p_n\}$ (where $s_{k+1} \neq s_j$ for all $j \leq k$) is selected and a new hypothesis $H_{s_{k+1}}$ is constructed by solving the single predicate learning task $spl(T, H_{s_{k+1}}, E_{s_{k+1}})$, where $E_{s_{k+1}}$ is the training set for predicate s_{k+1} and T is the theory $B \cup H_{cur}$. The background knowledge to be considered in single predicate learning is not only B , but B together with the m -hypothesis H_{cur} learned so far. We now show that this idea encounters some problems.

The main problem is to determine the next predicate to be learned. This is a serious problem because processing the predicates in a non-optimal order may significantly affect the learning results: no solution may be found or the induced hypothesis may be very complex.

The reason for the impact of the order of learning the predicates on the learning results is the use of a background theory. This theory determines the difficulty of the learning task. Given all relevant predicates in the background theory, the single predicate learning task is easy. On the other hand, if the available predicates are less relevant or irrelevant, the learning task becomes hard or unsolvable. For instance, learning the predicate `grandfather` in terms of `male`, `female` and `grandparent` is easy; using `male`, `female` and `parent` is harder, and using only `male` and `female` the problem is unsolvable.

The situation is more complicated than indicated above. All ILP (and other inductive) learners employ an explicit or implicit bias, which effectively restricts the space of considered hypotheses. Such a bias is necessary because the space of possible predicate definitions is the language $\mathcal{L}$ of definite or normal clause logic. Even for a fixed number of predicates of arity 2 or larger, the number of different clauses is infinite. Since no system can search forever, bias is needed to make the search space finite and to consider the most interesting clauses first. As examples of such bias, consider GOLEM's *ij*-determinacy (explicit declarative bias) and FOIL's encoding length restriction (implicit procedural bias). Bias makes the search space manageable, but can prevent the learner from finding the desired predicate definitions.

When trying to learn multiple predicates using a single predicate learner, bias and ordering effects are both to be taken into account. For certain orderings of learning the predicates, the learning task as a whole may be solvable, whereas for others, it may not. Clearly, in the worst case, which arises when there would exist only one order of learning the predicates that leads to a solution, all possible orderings of learning the predicates are to be considered (formulated more precisely, an ordering is a sequence of the learning tasks $spl(T, H_{p_i}, E_{p_i})$ where $T = B \cup H_{cur}$ for the current k -hypothesis H_{cur} and there is precisely one task for each predicate p_i). This is clearly computationally very expensive. Even when the worst case does not arise, using a non-optimal order may have negative effects on the results, in the sense that the induced hypotheses may be unnecessarily complex and thereby also less reliable and accurate. To illustrate

our point, consider GOLEM on the task of Example 1. GOLEM requires that both mother and father are learned before ancestor, because the recursive clause for ancestor using parent is not determinate.

Using a single predicate learner to solve a multiple predicate learning task is even more problematic than sketched above. Many of the problems are due to learning at the *local* instead of at the *global* level:

3.2.1 Overgeneralization

When an overly general clause c is learned for a predicate p_i and p_i has to be used in a clause for a predicate p_j , overgeneralization may prevent the learner from finding globally consistent clauses for p_j . The reason is that the decision to add clause c to the theory is made at a *local* level, based on the examples for p_i only. When using this predicate later, there are also global effects. One effect may be that p_j falls off the learnability cliff [Wrobel, 1988] (all clauses for p_j may be overly general, or may not satisfy the bias restrictions). As an example, consider again GOLEM on the task of Example 1, with the order of learning father, mother and ancestor. Assume that – due to too few negative examples for father – the learned definition for father is overly general (e.g., $\text{father}(X,Y) \leftarrow \text{parent}(X,Y)$). In this case the definition of ancestor is not determinate again. The example implies that overgeneralization of one predicate can prevent the learner from finding the definitions of other predicates. It is hard to see how global effects can be taken into account by single predicate learners, which make local decisions.

3.2.2 Mutually recursive predicates

Mutually recursive predicates¹ complicate the learning process because the learning of one (mutually recursive) predicate should be interleaved with the learning of the other ones. To learn multiple mutually recursive predicates, one should – in the worst case – not only consider all orderings of the (single predicate) learning tasks, but all sequences (i.e. single predicate learning tasks may have to be considered more than once).

4 Towards multiple predicate learning

The above analysis motivates a different approach, which discovers a good order of learning the predicates; interleaves the learning of the different predicates; recovers from overgeneralization; and takes into account global effects. We now present such an approach.

4.1 Refinement operators

In ILP, the search space is determined by the particular hypothesis language $\mathcal{L}$ used; $\mathcal{L}$ contains the clauses allowed in the hypotheses. The search space can then be structured using the notion of a *refinement operator* (cf. [Shapiro, 1983]).

Definition 11 Given a language $\mathcal{L}$ and a clause c , a *refinement operator* ρ is a mapping from $\mathcal{L}$ to $2^{\mathcal{L}}$ such that

¹Although mutually recursive predicates are not frequently needed, mutual recursion cannot always be removed (without introducing additional new predicates).

for all $c \in \mathcal{L}$: $\rho(c)$ is the set of most general specializations of c in $\mathcal{L}$.

The notion of specialization most often used in ILP is θ -subsumption as introduced by [Plotkin, 1970]. In most ILP systems, refinement of a clause of the form $\text{Head} \leftarrow \text{Body}$ is performed by applying a substitution on the clause or by adding a literal to the *Body* [Shapiro, 1983].

In the multiple predicate learning algorithm MPL (outlined in Section 4.2), instead of refining clauses of the form $\text{Head} \leftarrow \text{Body}$, the idea is to refine clauses c of the form $\leftarrow \text{Body}$ where the head of the clause is empty. A refinement operator can then perform two syntactic operations on a clause (see also [De Raedt and Bruynooghe, 1993]):

- *Body refinement operator* ρ_B conjunctively adds to *Body* a literal *Lit* with a predicate q_i occurring in *B* ($i = 1, \dots, l$) or a predicate p_i from the set of target predicates ($i = 1, \dots, m$).
- *Head refinement operator* ρ_H replaces the empty head by a literal with a predicate name p_i from the set of target predicates ($i = 1, \dots, m$).

All body refinements form a set $\rho_B(\text{Body}) = \{\text{Body} \wedge \text{Lit} \mid \text{Lit can be formed for a predicate } q_i (i = 1, \dots, l) \text{ or } p_i (i = 1, \dots, m)\}$. All head refinements performed on $\leftarrow \text{Body}$ form a set $\rho_H(\text{Body}) = \{\text{Head} \leftarrow \text{Body} \mid \text{literal Head can be formed for a predicate } p_i (i = 1, \dots, m)\}$. Head refinements, performed on $\rho_B(\text{Body})$, form a set $\rho_{HB}(\text{Body})$. Clauses constructed using a *B*, *H* or *HB* refinement operator will be called *B*-refinement, *H*-refinement or *HB*-refinement, respectively.

Our algorithm uses the head and body refinement operators to construct the clause to be added to the current hypothesis. Instead of refining a clause with a fixed predicate name in the head, the refinement operator generates clauses for different predicates.

4.2 The MPL algorithm

Here we outline a basic algorithm, named MPL (Multiple Predicate Learner), that incorporates the above ideas and that has proven capable of solving some multiple predicate learning tasks falling outside the scope of most of the other ILP systems.

The MPL system basically performs a hill-climbing search on m -hypotheses. In this part of the search, it keeps track of two hypotheses, H_{cur} and H_{next} . H_{cur} contains the current m -hypothesis, i.e. the best hypothesis encountered so far. H_{next} is constructed from H_{cur} in three steps :

First, clauses responsible for covering negative examples (globally incorrect clauses) are deleted. In case of different possibilities, preference is given to locally inconsistent clauses, and those clauses covering the most negative examples. Positive examples exclusively covered by retracted clauses are (again) added to the current example set. Incorrect deleted clauses are marked. The marks are used to avoid adding the same clause(s) later (cf. procedure Findbest).

Second, clauses that are no longer required to cover positive examples (globally irrelevant clauses)

are deleted. Clauses that previously covered positive examples may become irrelevant because of retractions/additions of other clauses.

Third, the algorithm repeatedly adds locally consistent clauses to H_{next} until a clause is added that is locally consistent but not globally or H_{next} is a solution to the multiple predicate learning task. Adding a clause that is not globally consistent leads to an inconsistency, which may be removed in the next cycle through the outer while loop.

If the m -hypothesis H_{next} generated in this way is better than H_{cur} (for a definition of better see Section 4.3), H_{next} is assigned to H_{cur} and the process is repeated.

The motivation for performing a greedy hill-climbing search is that it is far too expensive to investigate all alternatives. Furthermore, doing a complete search of the space of the alternatives cannot guarantee solutions as the search for clauses (cf. Findbest, which is similar to FOIL) is also heuristic.

MPL(B, E)

```

 $E_{cur} := E$ ;  $H_{next} := \emptyset$ ; better := true
while not( $H_{next}$  glob. comp. and cons.) and better do
   $H_{cur} := H_{next}$ ;
  while  $H_{next}$  glob. incons. do
    delete from  $H_{next}$  a glob. incons. clause  $c$ 
    add to  $E_{cur}$  all positive examples
      covered by  $c$  and not covered by  $H_{next}$ 
    delete from  $H_{next}$  all globally irrelevant clauses
  while  $H_{next}$  glob. cons. and not glob. comp. do
     $c_{best} := \text{call Findbest}(B, H_{next}, E_{cur})$ 
    add  $c_{best}$  to  $H_{next}$ 
    remove from  $E_{cur}$  all pos. ex. covered by  $H_{next}$ 
  better :=  $H_{next}$  is better than  $H_{cur}$ 
if  $H_{next}$  glob. complete and consistent
then delete from  $H_{next}$  all glob. irrelevant clauses
  output  $H_{next}$ 
else output  $H_{cur}$ 

```

Findbest(B, H, E)

```

Beam := { true }
while improvement possible do
  select best  $Body$  from Beam
  BRefinements :=  $\rho_B(Body)$ 
  for all  $Bo \in BRefinements$  do
    Evaluate  $Bo$ , i.e.
     $val(Bo) := \max\{val(c) \mid c \in \rho_H(Bo)\}$ 
  Beam :=  $b$  best bodies on Beam  $\cup$  BRefinements
return best unmarked clause on Beam

```

The search for locally consistent clauses (cf. Findbest) is based on the ideas in FOIL [Quinlan, 1990] and mFOIL [Džeroski, 1991]. At each step of the search, the algorithm keeps track of the b best bodies of clauses. It repeatedly selects the current best body from the Beam, and computes all B -refinements. The quality of a Bo is then taken as the maximum value of the evaluation function val (cf. Section 4.3) applied to the H -refinements of Bo . In this way, the predicate for which a clause is being learned is not fixed as in FOIL but rather selected dynamically. This allows to find the most interesting clauses first. The search continues as long as the bodies on Beam may still be improved. One further difference

with FOIL should be mentioned. The MPL algorithm marks the incorrect clauses it retracts. The algorithm takes care that marked clauses are never again added to current hypothesis. In this way, MPL avoids loops.

Notice that MPL can also start from an initial, approximate hypothesis for the predicates to be learned. The only required change is to initialize H_{cur} to this initial hypothesis.

4.3 Heuristics

The MPL sketched in Section 4.2 employs two heuristics: a heuristic to decide whether one m -hypothesis is better than another one, and a heuristic evaluation function that computes the quality (interestingness) of a clause.

Definition 12 Given a clause c for a predicate p_i , the current example set E_{cur} , the background theory B and the current hypothesis H_{cur} ,

$$val(c) = w_1 \times \frac{p_l + 1}{t_l + 2} + w_2 \times \frac{p_g + 1}{t_g + 2}$$

where p_l (resp. p_g) is the number of positive examples covered at the local (resp. global) level, and t_l (resp. t_g) the number of examples covered at the local (resp. global) level. Clause c_1 is better than clause c_2 if $val(c_1) \geq val(c_2)$. Ties are resolved by selecting the clause that contains the least number of literals involving $p_1, \dots, p_m$.

The evaluation function val is a weighted sum of the local (clause) and global (hypothesis) accuracy, estimated by the Laplace estimate also used in the ILP system mFOIL [Džeroski, 1991]. The tie-break procedure reflects a desire to have as few dependencies as possible, thereby reducing the probability of later revisions.

Finally, our algorithm also compares the quality of two m -hypotheses.

Definition 13 An m -hypothesis H_1 is better than H_2 if H_1 globally covers more positive examples than H_2 or H_1 covers the same number of positive examples as H_2 but less negative examples.

The algorithm uses a depth-bounded interpreter for testing coverage.

5 Experimental results

Using a prototype implementation of the MPL algorithm in PROLOG, two experiments were performed on the domain of learning family relations. In order to simplify the evaluation of the results and the comparison with FOIL, during all runs with the system, the system was not using the beam-search strategy but rather hill-climbing (the beam width b was 1). Furthermore, the depth bound for testing coverage was 6, the refinement operator being used was essentially the same as in Quinlan's FOIL, and the weights were $w_1 = 0.8$ and $w_2 = 0.2$.

Experiment 1 aimed at learning the definitions of ancestor, father and mother from the knowledge base containing male, female and parent specified in Example 1, and from a complete set of positive and negative examples. The system learned the following clauses in the specified order

- (1) $\text{ancestor}(X,Y) \leftarrow \text{parent}(X,Y)$
- (2) $\text{father}(X,Y) \leftarrow \text{parent}(X,Y) \wedge \text{male}(X)$
- (3) $\text{mother}(X,Y) \leftarrow \text{parent}(X,Y) \wedge \text{female}(X)$
- (4) $\text{ancestor}(X,Y) \leftarrow \text{parent}(X,Z) \wedge \text{ancestor}(Z,Y)$

To check the ability of the MPL algorithm to recover from overgeneralization, we also ran MPL starting from clauses 5, 6 and 7. This theory could have been induced using other evaluation criteria in MPL. Indeed after learning clause 5, ancestor and parent are synonyms and only our tie-break method prefers clause 2 over 6 and clause 3 over 7.

- (5) $\text{ancestor}(X,Y) \leftarrow \text{parent}(X,Y)$
- (6) $\text{father}(X,Y) \leftarrow \text{ancestor}(X,Y) \wedge \text{male}(X)$
- (7) $\text{mother}(X,Y) \leftarrow \text{ancestor}(X,Y) \wedge \text{female}(X)$
- (8) $\text{ancestor}(X,Y) \leftarrow \text{parent}(X,Z) \wedge \text{parent}(Z,Y)$

MPL then discovered that clause 8 was the most interesting one to add next, yielding the current hypothesis H composed of 5, 6, 7 and 8: At that moment, the system discovered that the hypothesis is no longer globally consistent and it decided to remove (and mark) clauses 6 and 7 from H . After this clauses 2, 3 and 4 were induced, making clause 8 redundant. Therefore clause 8 was removed, yielding the desired target theory.

Experiment 2 aimed at learning male_ancestor (m-a) and female_ancestor (f-a) from father and mother. All the facts and examples about father, mother, m-a and f-a were derived from the background theory in Example 1. MPL learned (notice that clause 14 is correct):

- (9) $\text{f-a}(X,Y) \leftarrow \text{mother}(X,Y)$
- (10) $\text{m-a}(X,Y) \leftarrow \text{father}(X,Y)$
- (11) $\text{f-a}(X,Y) \leftarrow \text{mother}(X,Z) \wedge \text{f-a}(Z,Y)$
- (12) $\text{m-a}(X,Y) \leftarrow \text{father}(X,Z) \wedge \text{f-a}(Z,Y)$
- (13) $\text{m-a}(X,Y) \leftarrow \text{father}(X,Z) \wedge \text{m-a}(Z,Y)$
- (14) $\text{f-a}(X,Y) \leftarrow \text{f-a}(X,Z) \wedge \text{m-a}(Z,Y)$

6 Conclusions

We analyzed the problem of empirically learning multiple predicates from examples. We identified the main problems: consideration of clauses only at a local level, extensional notion of coverage, determining the order of learning the predicates, bias, overgeneralization and mutual recursion. We also proposed an alternative approach MPL based on FOIL. Preliminary experiments show that MPL is able to solve some multiple predicate learning problems beyond the scope of other empirical ILP systems.

Acknowledgements

The authors are grateful to Hilde Ade, Francesco Bergadano, Peter Flach, Stephen Muggleton, and Gunther Sablon for discussions and/or comments. Special thanks to Maurice Bruynooghe for his support.

References

[De Raedt and Bruynooghe, 1993] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1993.

[De Raedt et al., 1993] L. De Raedt, N. Lavrač, and S. Džeroski. Multiple predicate learning. In *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, 1993.

[De Raedt, 1992] L. De Raedt. *Interactive Theory Revision: an Inductive Logic Programming Approach*. Academic Press, 1992.

[Džeroski, 1991] S. Džeroski. Handling noise in inductive logic programming. Master's thesis, University of Ljubljana, 1991.

[Kietz and Wrobel, 1992] J-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, 1992.

[Lavrač et al., 1991] N. Lavrač, S. Džeroski, and M. Grobelnik. Learning non-recursive definitions of relations with LINUS. In Yves Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1991.

[Lloyd, 1987] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 2nd edition, 1987.

[Muggleton and Buntine, 1988] S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–351. Morgan Kaufmann, 1988.

[Muggleton and Feng, 1990] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st conference on algorithmic learning theory*. Ohmsha, Tokyo, Japan, 1990.

[Muggleton, 1992] S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.

[Plotkin, 1970] G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5. Edinburgh University Press, 1970.

[Quinlan, 1990] J.R. Quinlan. Learning logical definition from relations. *Machine Learning*, 5:239–266, 1990.

[Richards and Mooney, 1991] B.L. Richards and R.J. Mooney. First order theory revision. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 447–451. Morgan Kaufmann, 1991.

[Shapiro, 1983] E.Y. Shapiro. *Algorithmic Program Debugging*. The MIT press, 1983.

[Wogulis, 1991] J. Wogulis. Revising relational theories. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 462–466. Morgan Kaufmann, 1991.

[Wrobel, 1988] S. Wrobel. Automatic representation adjustment in an observational discovery system. In Sleeman D., editor, *Proceedings of the 3rd European Working Session on Learning*. Pitman, 1988.