Integrating Extensional and Intensional ILP Systems through Abduction

E. Lamma, P. Mello, M. Milano, F. Riguzzi

1 DEIS, Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
{elamma,mmilano,friguzzi}@deis.unibo.it
Tel:+39 51 643033, Fax: +39 51 6443073

2 Dipartimento di Ingegneria, Università di Ferrara
Via Saragat 1, 41100 Ferrara, Italy
pmello@ing.unife.it

Abstract. We present an hybrid extensional-intensional Inductive Logic Programming algorithm. We then show how this algorithm solves the problem of global inconsistency of intensional systems when learning multiple predicates, without incurring in the problems of incompleteness and inconsistency of extensional systems. The algorithm is obtained by modifying an intensional system [6] for learning abductive logic programs. Extensionality is thus obtained by exploiting abduction: the training set is considered as a set of abduced literals that is taken as input by the abductive proof procedure used for the coverage of examples.

1 Introduction

Logic programs are rarely constituted by a single predicate. However, most Inductive Logic Programming (ILP) systems have been designed for learning single predicates. If we synthesize multiple predicates programs by applying single predicate learners, we find two problems [16]. The first is that adding a clause to a partial hypothesis can make previous clauses inconsistent. The second is that a very expensive backtracking on clause addition to the theory must be performed.

In order to overcome these problems, most top-down systems (e.g. ICN [13], MULT-ICN [12], FOIL [15], FOCL [14], MIS [17] with the lazy strategy) use extensional coverage: the coverage verification of examples is performed by using only the current clause, the background knowledge and the training set, but not previously learned clauses. In this way, clauses are learned independently from each other. We will distinguish between extensional and intensional systems depending on whether they use extensional coverage or not. Extensional coverage introduces other problems because the learning algorithm can be unsound: the learned theory can be both inconsistent and incomplete.

Therefore, we propose an hybrid system that employs both the training set and the current partial hypothesis in the derivation of examples. In this way, we solve the problem of the consistency of partial hypothesis in intensional systems, without incurring in the most significant problems of extensional systems.
The algorithm we propose is obtained by modifying the one presented in [6] for learning abductive logic programs. The problem of learning abductive logic programs is emerging as a promising research direction in the field of ILP. A number of works [6, 9, 11, 10] have started to appear on the subject, and, more generally, on the relation existing between abduction and induction and how they can integrate and complement each other [4, 5, 1].

We exploit abduction in order to introduce extensionality: the training set is considered as a set of abduced literals that is taken as input by the abductive proof procedure used for deriving examples. The abduced literals are considered as additional facts that are true in the theory. The resulting algorithm is able to learn abductive logic programs without incurring in the problem of inconsistent hypotheses and, as a special case, is also able to correctly learn standard logic programs.

The paper is organized as follows: in section 2 we present the various problems of intensional and extensional systems and how they are solved by using an hybrid algorithm. In section 3 we describe the integration of abduction and induction that was proposed in [6]. In section 4 we present the abductive inductive algorithm modified in order to perform hybrid coverage. In section 5 we conclude and present the directions for future works.

2 Extensional and Intensional Systems

Most top-down ILP systems use extensional coverage because of two problems of intensionality: the cost of backtracking on clause addition and the difference between local and global consistency when learning multiple predicates. These two problems have been discussed in [16] and are reported below.

With a great deal of approximation, top-down systems share a common basic algorithm [2]:

\[
T := \emptyset \\
\textbf{while } E^+ \neq \emptyset \textbf{ do (Covering loop)} \\
\quad \text{Generate one clause } C \\
\quad \text{Remove from } E^+ \text{ the } e^+ \text{ covered by } C \\
\quad \text{Add } C \text{ to } T \\
\text{Generate one clause } C \text{ (specializing loop):} \\
\quad \text{Select a predicate } p \text{ that must be learned} \\
\quad \text{Set clause } C \text{ to be } p(X) \leftarrow . \\
\textbf{while } C \text{ covers some negative example do} \\
\quad \text{Select a literal } L \text{ from the language bias} \\
\quad \text{Add } L \text{ to the body of } C \\
\quad \text{Test coverage of } C \\
\quad \text{if } C \text{ does not cover any positive example} \\
\qquad \text{then backtrack to different choices for } L \\
\textbf{return } C \\
\text{(or fail if backtracking exhausts all choices for } L) 
\]
This algorithm iteratively adds a clause to the current partial theory. It generates a clause by searching depth-first the space of possible clauses. Added clauses are never retracted because backtracking on clause addition would be too expensive, therefore the search in the space of possible programs is greedy. However, we may add to the theory a certain number of clauses and then find out that no other clause is available in the language bias for covering the remaining positive examples, while with a different choice of previous clauses we could have had a solution. This problem arises both when learning single recursive predicates, because clauses are dependent on each other, and when learning multiple predicates. However, when learning multiple predicates the problem is more evident because the dependency relations between different clauses are more frequent and complex. By using extensional coverage we solve this problem because we do not need anymore to search the space of possible programs, it is sufficient to iteratively search the smaller space of possible clauses since the evaluation of a clause is independent from the clauses already learned.

**Definition 1 Extensional coverage.** The clause \( c = t \leftarrow l_1, l_2 \ldots l_n \) extensionally covers \( e \) iff \( \theta \) unifies with \( c \) with substitution \( \theta \) and \( B \cup E^+ \models [t_1, l_2 \ldots l_n] \theta \).

The second problem appears when learning multiple predicates. In this case, we have to distinguish between two types of consistency: relative local and relative global consistency of a new clause with respect to the theory learned so far (hypothesis). These definitions are based on the absolute definition of local and global consistency of a clause given in [16]. Let us first give some terminology and define the function \( \text{covers}(B, H, E) \) [16]. We assume that the training sets \( E^+ \) and \( E^- \) contain examples for \( m \) predicates \( p_1, \ldots, p_m \) and we partition \( E^+ \) and \( E^- \) in \( E^+_{p_i} \) and \( E^-_{p_i} \) according to these predicates.

**Definition 2** \( \text{covers}(B, H, E) \). Given the background theory \( B \), the hypothesis \( H \) and the example set \( E \), \( \text{covers}(B, H, E) = \{ e \in E \mid B \cup H \models e \} \).

**Definition 3** Relative global consistency. Given a clause \( c \) for the predicate \( p_i \) and the consistent hypothesis \( H \), the clause \( c \) is globally consistent with respect to \( H \) iff \( \text{covers}(B, H \cup c, E^-_{p_i}) = \emptyset \).

**Definition 4** Relative local consistency. Given a clause \( c \) for the predicate \( p_i \) and the consistent hypothesis \( H \), the clause \( c \) is locally consistent with respect to \( H \) iff \( \text{covers}(B, H \cup c, E^-) = \emptyset \).

The basic top-down algorithm has been designed for learning single predicates: it generates a theory by iteratively adding a relatively locally consistent clause to the current partial theory. However, when learning multiple predicates, adding a relatively locally consistent clause to a consistent hypothesis can produce an inconsistent hypothesis as it is shown in the next example.

**Example 1.** Suppose we want to learn the predicates \( p \) and \( q \) from the background knowledge and the training sets

\[
B = \{ a(1) \leftarrow, \quad b(2) \leftarrow \}
\]
Suppose also the system first learns the rule \( p(X) \leftarrow a(X) \) which covers \( p(1) \) and does not cover \( p(3) \), then learns the rule \( q(X) \leftarrow p(X) \) which covers \( q(1) \) and does not cover \( q(2) \) because the current definition for \( p \) covers only \( p(1) \).

Then the rule \( p(X) \leftarrow b(X) \) is learned which covers \( p(2) \) and now the second rule becomes inconsistent because \( q(2) \) is covered.

Therefore, in intensional systems, it is not enough to check the local consistency of a clause, but the global consistency must be checked, as it is done in the system MPL [16]. This is equivalent to test the coverage of all the negative examples, that can be computationally very expensive.

Also this second problem can be solved less expensively by using extensional coverage. With extensional coverage we can test only the extensional local consistency and be sure that the clause is also extensionally globally consistent. By using extensional coverage in example 1 the second rule would not be consistent because the negative example \( q(3) \) would be extensionally covered using the positive example \( p(2) \) for \( p \).

However, extensional coverage poses a number of other problems, due to the fact that the learned theory is tested differently from the way in which it is effectively used. In particular, we can have the following cases [16]: (i) extensional consistency, intensional inconsistency; (ii) extensional completeness, intensional incompleteness; (iii) intensional completeness, extensional incompleteness.

In order to avoid the problems of extensional systems and the problem of global inconsistency of intensional systems, we resort to an hybrid system, as done in [7]. In this way, we get a system that, when trying to derive an example using the last generated rule, first checks if the literals in the body of the rule are in the set of positive or negative examples and, if no example matches, starts a derivation for the literals using the current hypothesis and the background knowledge.

This system however still presents the problem of extensional completeness, intensional incompleteness of extensional systems. For example, we may learn a recursive program without a base clause:

\[
\begin{align*}
\text{even}(X) & \leftarrow \text{succ}(X, Y), \text{odd}(Y). \\
\text{odd}(X) & \leftarrow \text{succ}(X, Y), \text{even}(Y).
\end{align*}
\]

because the examples are used as base clauses. A solution to this problem has been proposed in [12] with the system MULTJCN.

Differently from [7], instead of building an ad hoc system, we rely on abduction in order to address extensional coverage and integrate extensional and intensional behaviour in ILP. To this purpose, we exploit the ILP system for learning abductive logic programs defined in [6]: the coverage of examples is done employing both the current partial hypothesis and the training sets viewed as abductive hypothesis.
3 Integrating Abductive and Inductive Logic Programming

In this section, we recall the approach for the integration of abduction and induction that was proposed in [6, 11]. First, we summarize the main concepts of Abductive Logic Programming (ALP) and then we show how the learning problem of ILP must be modified in order to integrate abduction.

An abductive logic program is a triple \( \langle P, A, IC \rangle \) where \( P \) is a normal logic program, \( A \) is a set of abducible predicates, \( IC \) is a set of integrity constraints in the form of denials, i.e.: \( \leftarrow A_1, \ldots, A_m, \neg \neg A_{m+1}, \ldots, \neg \neg A_{m+n} \).

Negation as Failure is replaced, in ALP, by Negation by Default and is obtained in this way: for each predicate symbol \( p \), a new predicate symbol \( \neg \neg p \) is added to the set \( A \) and the integrity constraint \( \leftarrow p(X) \), \( \neg \neg \neg p(X) \) is added to \( IC \), where \( X \) is a tuple of variables.

Given an abductive program \( AT = \langle P, A, IC \rangle \) and a formula \( G \), the goal of abduction is to find a (possibly minimal) set of ground atoms \( \Delta \) (abductive explanation) of predicates in \( A \) which together with \( P \) entails \( G \), i.e. \( P \cup \Delta \models G \). It is also required that the program \( P \cup \Delta \) is consistent with respect to \( IC \), i.e. \( P \cup \Delta \models IC \). We say that \( AT \) abductively entails \( e \) (\( AT \models_{A} e \)) when there exist an abductive explanation for \( e \) from \( AT \). We adopt the three-valued semantics for ALP defined in [3] in which an atom can be true, false or unknown. In this way we can model domains where the knowledge is not complete.

In [8] a proof procedure for abductive logic programs has been defined. This procedure starts from a goal and a set of abduced literals \( \Delta_{in} \) and results in a set of consistent assumptions \( \Delta_{out} \) (abduced literals) such that \( \Delta_{out} \subseteq \Delta_{in} \) and \( \Delta_{out} \) together with the program allow to derive the goal. We write

\[ AT \models_{\Delta_{out}} \Delta_{in} G \]

We have extended this proof procedure in order to allow for abducible predicates to have a partial definition. Some rules may be available for them, and we can make assumptions about missing facts.

We consider a new definition of the learning problem similar to Abductive Concept Learning (ACL) [5]. In this extended learning problem both the background and target theory are abductive theories and the notion of deductive entailment is replaced by abductive entailment.

**Given**
- a set \( \mathcal{P} \) of possible abductive programs
- a set of positive examples \( E^+ \),
- a set of negative examples \( E^- \),
- an abductive theory \( AT = \langle T, A, IC \rangle \) as background theory.

**Find**
- A new abductive theory \( AT' = \langle T', A, IC \rangle \in \mathcal{P} \) with \( T' \supseteq T \), such that
  \[ \forall e \in E, \quad AT' \models_{\Delta_{in}} e \]
  \[ \forall e \in \neg \neg \neg E, \quad AT' \models_{\Delta_{out}} e \]
When $\forall \varepsilon \in E$ we say that $\forall \varepsilon \text{ abductively covers } e$.

The abductive program that is learned can contain new rules (possibly with abducibles in the body) but not new abducible predicates and new integrity constraints.

In order to introduce extensional coverage in this framework, we have to change the condition that the learned program must satisfy:

$$\forall \varepsilon \in E, \forall t \in b [\varepsilon \cup T \models I]$$

Differently from def. 1 for extensional coverage, here also negative examples are used because of the three-valued semantics of abduction [3]. The literal $l$ is proved true if $l \in E$ and is proved false if $l \notin E$ where

$$T = \begin{cases} n_{ol}(X) & \text{if } l = p(X) \\ p(X) & \text{if } l = n_{ol}(X) \end{cases}$$

**4 The hybrid algorithm**

We present an intensional algorithm that is able to learn abductive logic programs [6] and we show how it can be extended, by exploiting abduction, to incorporate extensional coverage. The algorithm is obtained from the basic top-down ILP algorithm [2], by substituting the usual notion of coverage of examples with the notion of abductive coverage.

The basic top-down algorithm is extended in the following respects in order to learn abductive logic programs. First, in order to test the coverage of the generated rule, an abductive derivation is started for each positive example and the default negation ($\not\neg e^+$) of each negative ($e^-$). Each derivation starts from the set of literals abduced in the derivations of the previously covered examples. In this way, we ensure that the assumptions made during the derivation of the current example (positive or negative) are consistent with the assumptions previously raised for deriving other examples.

Second, after the generation of each clause, the abduced literals of target predicates are added to the training set, so that they become new training examples.

In order to introduce extensional coverage in the algorithm, each abductive derivation of an example starts not only with the set of literals already abduced but also with the training set itself. In particular, the input abducibles are augmented with all the positive examples and the default negation of each negative. In order to avoid the trivial derivation of $e^+$ based on $e^+$ itself, $e^+$ is taken out from the input abducibles. The same is done for $\not\neg e^-$.

We now show an example of the behaviour of the algorithm in the case of learning the predicate member. Let the background knowledge and training set be:
B = \{\text{components(}[H,T],[H,T]) \}</p>

E^+ = \{\text{member}(2, [2]), \text{member}(2, [1, 2, 3]), \text{member}(3, [1, 2, 3])\}

E^- = \{\text{member}(2, [1]), \text{member}(2, [3]), \text{member}(1, [2, 3])\}

Suppose the system first generates the clause

\text{member}(A, B) \leftarrow \text{components}(B, C, D), \text{member}(A, D)

Then the clause is tested. The abductive derivation \(\text{member}(2, [2])\) fails because \(\text{member}(2, [2])\) can not be derived nor abduced, since it is a negative example. In the abductive derivation of \(\text{member}(2, [1, 2, 3])\), first the system unfolds two times the clause and tries to abduce \(\text{member}(2, [3])\). Since it is a negative example, the derivation fails and, in backtracking, it succeeds with the abduction of \(\text{member}(2, [2, 3])\). Finally, the positive example \(\text{member}(3, [1, 2, 3])\) is covered with the abduction of \(\text{member}(3, [3])\). Then negative examples are tested: \(\neg \text{member}(2, [1])\), \(\neg \text{member}(2, [3])\) and \(\neg \text{member}(1, [2, 3])\) all succeeds. In the last case, \(\neg \text{member}(1, [3])\) is abduced. Therefore the rule is consistent and is added to the hypothesis. Covered positive examples are removed and assumptions about target predicates are added to the training set, that becomes:

\begin{align*}
E^+ &= \{\text{member}(2, [2]), \text{member}(2, [2, 3]), \text{member}(3, [3])\} \\
E^- &= \{\text{member}(2, [1]), \text{member}(2, [3]), \text{member}(1, [2, 3]), \text{member}(1, [3])\}
\end{align*}

Then the system generates the clause

\text{member}(A, B) \leftarrow \text{components}(B, A, D)

that covers all the remaining positive examples and the negation of the negative ones without abducing anything. The clause is added to the hypothesis and the algorithm terminates.

5 Conclusions and Future Work

We have shown how abduction can be used in order to introduce extensionality in intensional systems. In particular, we have taken the intensional system for learning abductive logic programs proposed in [6] and we have extended it in order to include extensional coverage. In this way, we get an hybrid system that overcomes the problem of global inconsistency of intensional systems without incurring in the most significant problems of completeness and consistency of extensional systems.

Systems related to ours are MPL [16] and FOL-I [7]. MPL is an intensional system that solves the problem of global inconsistency when learning multiple predicates by checking the global consistency of the hypothesis, while FOL-I introduces intensional coverage in an extensional system.

In the future, we will consider how abduction can be used in order to solve the problems of intensional systems when learning normal logic programs. In this case, adding a clause to a partial hypothesis can reduce the coverage of previous clauses. This makes most intensional systems unable to learn normal logic programs because covered positive examples can, at a later stage, be uncovered again.
References


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