Learning recursive theories with the separate-and-parallel-conquer strategy

Antonio Varlaro Margherita Berardi Donato Malerba

Dipartimento di Informatica – Università degli Studi di Bari
via Orabona 4 - 70126 Bari
{varlaro, berardi, malerba}@di.uniba.it

Abstract. Inductive learning of recursive logical theories from a set of examples is a complex task characterized by three important issues that do not arise in single predicate learning, namely the adoption of a generality order stronger than θ-subsumption, the non-monotonicity of the consistency property, and the automated discovery of dependencies between target predicates. Some variations of the classical separate-and-conquer strategy to overcome these issues have been implemented in the learning system ATRE and are reported in this paper. An application to the document image understanding domain is discussed and conclusions are drawn.

1. Introduction

Inductive Logic Programming (ILP) is a research area at the intersection of Machine Learning and Logic Programming. ILP systems induce predicate descriptions that together with a background knowledge explain some properties of a set of observations (examples). The examples, background knowledge and final descriptions are all described as logic programs. Typically, the output of an ILP system is a logical theory expressed as a set of definite clauses (clauses with exactly one literal in their head), which logically entail all positive examples and no negative example. Therefore, each concept definition corresponds to a predicate definition and a concept learning problem is reformulated as a predicate learning problem.

In this paper we are interested in learning multiple predicate definitions (or recursive theories), provided that both positive and negative examples of each concept/predicate to be learned are available. Multiple predicate learning is a more complex case of the classical ILP problem of single predicate learning from a set of positive and negative examples. Complexity stems from the fact that the learned predicates may also occur in the antecedents of the learned clauses, that is, the learned predicate definitions may be interrelated and depend on one another, either hierarchically or involving some kind of mutual recursion. For instance, to learn the definitions of odd and even numbers, a multiple predicate learning system will be provided with positive and negative examples of both odd and even numbers, and may generate the following recursive logical theory:

$$\text{odd}(X) \leftarrow \text{succ}(Y,X), \text{even}(Y)$$

$$even(X) \leftarrow succ(Y,X), odd(Y)$$
$$even(X) \leftarrow zero(X)$$

where the definitions of *odd* and *even* are interdependent. This example shows that the problem of learning multiple predicate definitions is equivalent, in its most general formulation, to the problem of learning recursive logical theories.

There has been considerable debate on the actual usefulness of learning recursive logical theories in knowledge acquisition and discovery applications. This is probably due to the fact that very few real life concepts seem to have recursive definitions, rare examples being "ancestor" and natural language [1, 9]. However, in the literature it is possible to find several ILP applications in which recursion has proved helpful [6]. Moreover, many ILP researchers have shown some interest in multiple predicate learning [5], which presents the same difficulty of recursive theory learning in its most general formulation.

To formulate the recursive theory learning problem and then to explain its main theoretical issues, some basic definitions are given below.

Generally, every logical theory T can be associated with a directed graph $\gamma(T)=<N,E>$, called the *dependency graph* of T, in which (i) each predicate of T is a node in N and (ii) there is an arc in E directed from a node a to a node b, iff there exists a clause C in T, such that a and b are the predicates of a literal occurring in the head and in the body of C, respectively. The dependency graph represents the predicate dependencies of T, where a *predicate dependency* is defined as follows:

Definition 1 (predicate dependency). A predicate p depends on a predicate q in a theory T iff (i) there exists a clause C for p in T such that q occurs in the body of C; or (ii) there exists a clause C for p in T with some predicate r in the body of C that depends on q.

Definition 2 (recursive theory). A logical theory T is *recursive* if the dependency graph $\gamma(T)$ contains at least one cycle.

In *simple* recursive theories all cycles in the dependency graph go from a predicate p into p itself, that is, simple recursive theories may contain recursive clauses, but cannot express mutual recursion.

Definition 3 (predicate definition). Let T be a logical theory and p a predicate symbol. Then the *definition* of p in T is the set of clauses in T that have p in their head. Henceforth, $\delta(T)$ will denote the set of predicates defined in T and $\pi(T)$ will denote the set of predicates occurring in T, then $\delta(T) \subseteq \pi(T)$.

In a quite general formulation, the recursive theory learning task can be defined as follows:

Given

- A set of *target* predicates $p_1, p_2, \ldots, p_r$ to be learned
- A set of positive (negative) examples E_i^+ (E_i^-) for each predicate p_i, $1 \le i \le r$
- A background theory BK
- A language of hypotheses $\mathcal{L}_H$ that defines the space of hypotheses S_H

Find

a (possibly recursive) logical theory $T \in S_H$ defining the predicates $p_1, p_2, \ldots, p_r$ (that is, $\delta(T)=\{p_1, p_2, \ldots, p_r\}$) such that for each i, $1 \le i \le r$, $BK \cup T \models E_i^+$ (*completeness* property) and $BK \cup T \not\models E_i^-$ (*consistency* property).

Three important issues characterize recursive theory learning. First, the generality order typically used in ILP, namely θ-subsumption [11], is not sufficient to guarantee the completeness and consistency of learned definitions, with respect to logical entailment [10]. Therefore, it is necessary to consider a stronger generality order, which is consistent with the logical entailment for the class of recursive logical theories we take into account.

Second, whenever two individual clauses are consistent in the data, their conjunction need not be consistent in the same data [4]. This is called the *non-monotonicity property* of the normal ILP setting, since it states that adding new clauses to a theory T does not preserve consistency. Indeed, adding definite clauses to a definite program (i.e. a set of definite clauses) enlarges its least Herbrand model (LHM), i.e. the set of ground facts that are logically entailed by the theory, which may then cover negative examples as well. Because of this non-monotonicity property, learning a recursive theory one clause at a time is not straightforward.

Third, when multiple predicate definitions have to be learned, it is crucial to discover dependencies between predicates. Therefore, the classical learning strategy that focuses on a predicate definition at a time is not appropriate.

To overcome these problems some solutions have been proposed in [7] and implemented in the learning system ATRE (http://www.di.uniba.it/~malerba/software/atre). This approach differs from related works on multiple predicate learning, as clearly discussed in [7], for at least one of the following three aspects: the learning strategy, the generalization model, and the strategy to recover the consistency property of the learned theory when a new clause is added.

In this paper we focus on the main problem of the interleaving of the learning of one (possible recursive) predicate definition with the learning of the other ones. In particular, different aspects of the adopted strategy for the automated discovery of predicate dependencies, namely the separate-and-parallel-conquer strategy, as variations on the classical separate-and-conquer search strategy, are presented.

The paper is organized as follows. Section 2 illustrates details on the learning strategy adopted in ATRE to learn recursive theories. The application of ATRE to a real-world problem, namely document image understanding, is presented in Section 3. Finally, some conclusions are drawn.

2. The adopted learning strategy

The learning strategy adopted by ATRE to induce logical theories belongs to the large family of *separate-and-conquer* strategies. Indeed, ATRE basically works by learning one clause at a time and removing covered training examples from the whole set of examples. In this way, a recursive theory T is built step by step, starting from an empty theory T_0, and adding a new clause at each step. Thus, the final logical theory is incrementally built until all positive examples in the training set are covered:

$$T_0 = \varnothing, T_1, \ldots, T_i, T_{i+1}, \ldots, T_n = T$$

In the above sequence, for each i, $T_{i+1} = T_i \cup \{C\}$ holds (where C is the added clause), and all theories are complete and consistent with respect to the training set. Let *pos(T)* and *neg(T)* be the number of positive and negative examples covered by T,

respectively. If we guarantee the following two conditions: for each $i \in \{1,..,n\}$ $pos(T_{i-1}) < pos(T_i)$ and $neg(T_i)=0$, we ensure that, after a finite number of steps, the learning process terminates producing a complete and consistent theory.

If we denote with *LHM(T)* the least Herbrand model of a theory *T*, the separate-and-conquer learning strategy augments at each step the *LHM(T)* by adding a clause. Moreover, considering that only definite clauses are used in ATRE, the least Herbrand model can only augment at each step because only positive information are added to the model. Using this monotonicity property, the termination criterion is reached when the LHM(T) is great enough to contain the whole positive information of the training dataset.

In order to guarantee the first condition above, namely $pos(T_{i-1}) < pos(T_i)$, the following procedure is adopted. First, an uncovered positive example e^+ is selected from the uncovered training set; this example is called *seed*. Then the space of definite clauses more general than e^+ is explored, looking for a clause (if any) such that

$neg(T_{i-1} \cup \{C\})=0$ holds. This constraint guarantees that the second condition above holds as well. When found, *C* is added to T_{i-1} producing T_i and the new logical theory is matched against the training examples. All the covered positive examples are "separated" by the other ones and the procedure restarts. The learning task stops when there are no further seeds to choose.

2.1 A first variation of the separate-and-conquer strategy

The learning strategy showed in the previous section is suitable to learn simple predicate definitions. Some variations of the classical approach are worth to be explored in order to extend it to multiple predicate learning.

First of all, the *non-monotonicity property* of the normal ILP setting should be faced: whenever a consistent clause *C* is added to a consistent theory *T*, the resulting theory *T'* may not be consistent with respect to the whole training set. This can be explained with an example.

Example 1. Consider the problem of learning the definitions of *ancestor* and *father* from a complete set of positive and negative examples. Suppose that the following recursive theory T_2 has been learned at the second step:

C_1: *ancestor*(*X,Y*) ← *parent(X,Y)*

C_2: *father*(*Z,W*) ←. *ancestor(Z,W), male(Z)*

Note that T_2 is consistent but still incomplete. Thus a new clause will be generated at the third step of the sequential-covering strategy. It may happen that the generated clause is the following:

C: *ancestor*(*A,B*) ←. *parent(A,D),ancestor(D,B)*

which is consistent given T_2, but when added to the recursive theory, it makes clause C_2 inconsistent.

To remove this inconsistency, a variation of the classical separate-and-conquer learning strategy has been proposed. It consists in adding a further step at the end of the learning process whenever a theory inconsistency raises. In particular, if the

addition of a clause to a theory T makes it inconsistent, some simple syntactic changes are performed in T in order to eventually create new *layers*.

More precisely, the layering of a theory T is a partition of the set of clauses in T into n disjoint sets of clauses or *layers* T^i such that $LHM(T) = LHM(LHM(\cup_{j=0,\ldots,n-2}\ T^j)\cup T^{n-1})$, that is, $LHM(T)$ can be computed by iteratively applying the immediate consequence operator (i.e. an operatively way to compute the set of logical consequences of an interpretation) to T^i, starting from the interpretation $LHM(\cup_{j=0,\ldots,i-1}\ T^j)$, for each $i\in\{1, \ldots,n\}$. In [7] an efficient method for the computation of a layering is reported. It is based on the concept of *collapsed dependency graph* and returns a unique layering for a given logical theory T. The layering of a theory provides a semi-naive way of computing the generalized implication test, the generality order used in ATRE and presented below and provides a solution to the problem of consistency recovering when the addition of a clause makes the theory inconsistent.

Theorem 1. Let $T=T^0 \cup \ldots \cup T^i \cup \ldots \cup T^{n-1}$ be a consistent theory partitioned into n layers, and C be a definite clause whose addition to the theory T makes a clause in layer T^i inconsistent. Let $p\in\{p_1, p_2, \ldots, p_r\}$ be the predicate in the head of C. Let T'' be a theory obtained from T by substituting all occurrences of p in T with a new predicate symbol, p', and $T'=T''\cup\{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\}\cup\{C\}$. Then T' is consistent and $LHM(T) \subseteq LHM(T') \setminus \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\}$.

In short, the new theory T' obtained by renaming the predicate p with a new predicate name p' before adding C is consistent and keeps the original coverage of T. This introduces a first variation of the classical separate-and-conquer strategy sketched above, since the addition of a locally consistent clause C generated in the conquer stage is preceded by a global consistency check. If the result is negative, the partially learned theory is first restructured, and then two clauses, C and $p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)$, are added. For instance, in the example above the result will be:

C_1': *ancestor'*$(X,Y) \leftarrow$ *parent*(X,Y)
C_2': *father*$(Z,W) \leftarrow$ *ancestor'*(Z,W), *male*(Z)
ancestor$(U,V) \leftarrow$ *ancestor'*(U,V)
C: *ancestor*$(A,B) \leftarrow$ *parent*(A,D),*ancestor*(D,B)

It is noteworthy that, in the proposed approach to consistency recovery, new predicates are invented, which aim to accommodate previously acquired knowledge (theory) with the currently generated hypothesis (clause).

2.2 A second variation of the separate-and-conquer strategy

Another issue that should be taken into account in multiple predicate learning is the automated discovery of dependencies between target predicates. Indeed, in multiple predicate learning, target predicate definitions generated at the i-th step may include some target predicates for which at least a clause has been added to the partial theory learned in the previous step. In addition, the order according to which clauses of

distinct predicate definitions have to be generated is not known in advance, so it is necessary to generate clauses with different predicates in the head and then to pick one of them at the end of each step of the separate-and-conquer strategy. This leads to another variation of the classical separate-and-conquer learning strategy, namely *separate-and-parallel-conquer* learning strategy. In the proposed approach, the learning of each predicate definition is interleaved with the learning of the other predicate definitions in order to automatically discover predicate dependencies. In particular, since the generation of a clause depends on the chosen seed, several seeds have to be selected such that at least one seed per incomplete predicate definition is kept. Therefore, the search space is actually a forest of as many search-trees (called *specialization hierarchies*) as the number of chosen seeds. Operatively, the (downward) refinement operator considered in this work adds a new literal to a clause.

The interleaving of each predicate learning means that the forest is processed in parallel by as many concurrent tasks as the number of search-trees. Each task traverses the specialization hierarchy top-down (or general-to-specific), but synchronizes traversal with the other tasks at each level. Initially, some clauses at depth one in the forest are examined concurrently. Each task is actually free to adopt its own search strategy, and to decide which clauses are worth to be tested. If none of the tested clauses is consistent, clauses at depth two are considered. Search proceeds towards deeper and deeper levels of the specialization hierarchies until at least a user-defined number of consistent clauses is found. Task synchronization is performed after that all "relevant" clauses at the same depth have been examined. A supervisor task decides whether the search should carry on or not on the basis of the results returned by the concurrent tasks. When the search is stopped, the supervisor selects the "best" consistent clause according to the user's preference criterion. This strategy has the advantage that simpler consistent clauses are found first, independently of the predicates to be learned.

The parallel exploration of the specialization hierarchies for *odd* and *even* is shown in Fig. 1. In particular, a seed example is chosen for each concept to be learned, suppose *even(0)* and *odd(1)* and, starting from this selection, the specialization hierarchies are built. As first step, the hierarchies are rooted with the clauses *even(X)* ← and *odd(X)* ← to learn the *even* and *odd* definitions, respectively. Starting from the roots, a refinement step is performed level by level on available clauses by adding a literal. By exploring the two hierarchies, ATRE finds several candidate clauses to add to the initial empty theory. In this way, the simplest clause (i.e. *even(X)* ← *zero(X)*) is found first and is added to the theory as base case of recursion. To generate the second clause, two new seeds are considered (i.e. *even(2)* and *odd(1)*). The two consistent clauses (i.e. those in italics in the level 2 of the second *odd* hierarchy) will be evaluated and one of them will be added to the theory. It is noteworthy that the second one, namely *odd(X)* ← *succ(Y,X), even(Y)*, has been generated since a partial definition of the *even* concept has already been learned in the previous step.

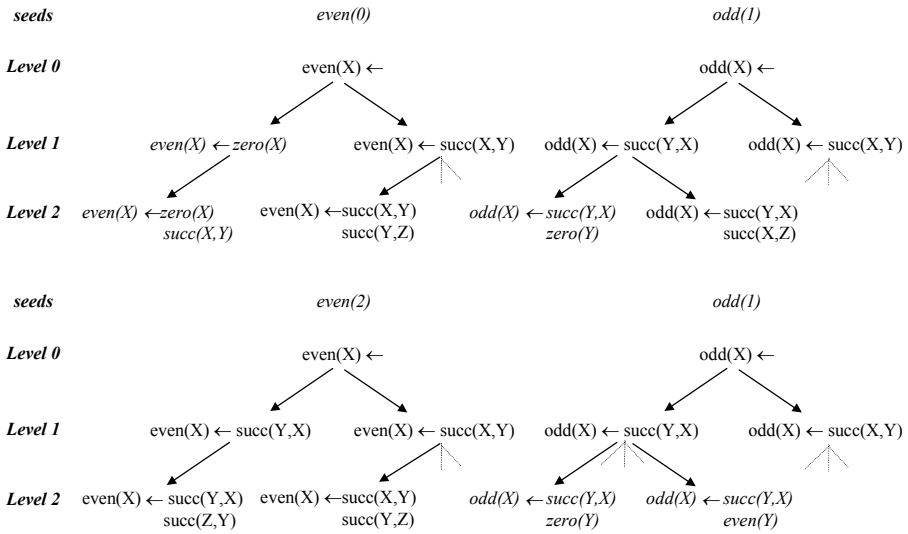

Fig. 1. Two steps (up and down) of the parallel search for the predicates *odd* and *even*. Consistent clauses are reported in italics.

2.3 A proper rule ordering

As explained in Section 1, in recursive theory learning it is necessary to consider a generality order that is consistent with the logical entailment for the class of recursive logical theories. A generality order (or *generalization model*) provides a basis for organizing the search space and is essential to understand how the search strategy proceeds. The main problem with the well-known θ-subsumption is that the objects of comparison are two clauses, say *C* and *D*, and no additional source of knowledge (e.g., a theory *T*) is considered. For instance, with reference to the previous example on *odd* and *even* predicates, the clause:

C: *odd(X)* ← *succ(Y,X)*, *even(Y)*

logically entails, and hence can be correctly considered more general than

D: *odd(3)* ← *succ(0,1)*, *succ(1,2)*, *succ(2,3)*, *even(0)*

only if we take into account the theory

T: *even(A)* ← *succ(B,A)*, *odd(B)*

even(C) ← *zero(C)*

Therefore, we are only interested in those generality orders that compare two clauses relatively to a given theory *T*, such as Buntine's *generalized subsumption* [2] and Plotkin's notion of *relative generalization* [11, 12].

Informally, generalized subsumption ($\leq_T$) requires that the heads of *C* and *D* refer to the same predicate, and that the body of *D* can be used, together with the background theory *T*, to entail the body of *C*. Unfortunately, generalized subsumption is too weak for recursive theories, because in some cases, given two clauses C and D, it may happen that $T \cup \{C\} \models D$ holds but it can not be concluded that $C \leq_T D$.

Plotkin's notion of *relative generalization* [11, 12] was originally proposed for a theory T of unit clauses. Buntine [2] reports an extension of relative generalization to the case of a theory T composed of definite clauses (not necessarily of unit clauses)

Definition 4 (relative generalization). Let C and D be two definite clauses. C is *more general than D* under relative generalization, with respect to a theory T, if a substitution θ exists such that $T \models \forall(C\theta \Rightarrow D)$.

The following theorem holds for this extended notion of relative generalization:

Theorem 2. Let C and D be two definite clauses and T a logical theory. C is *more general than D* under relative generalization, with respect to a theory T, if and only if C occurs at most once in some refutation demonstrating $T \models \forall(C \Rightarrow D)$.

However, this extended notion of relative generalization is still inadequate. From one side, it is still weak. Indeed, if we consider the clauses and the theory reported in the example above, it is clear that a refutation demonstrating $T \models \forall(C \Rightarrow D)$ involves twice the clause C to prove both *odd(1)* and *odd(3)*.

Malerba [7] has defined the following generalization order, which proved suitable for recursive theories.

Definition 5 (generalized implication). Let C and D be two definite clauses. C is *more general than D* under generalized implication, with respect to a theory T, denoted as $C \leq_{T\Rightarrow} D$, if a substitution θ exists such that $head(C)\theta = head(D)$ and $T \models \forall(C \Rightarrow D)$.

Decidability of the generalized implication test is guaranteed in the case of Datalog clauses [3]. In fact, the restriction to function-free clauses is common in ILP systems, such as ATRE, which remove function symbols from clauses and put them in the background knowledge by techniques such as flattening [13].

2.4 Some peculiarities of the learning strategy

Some points of the presented learning strategy have been left unspecified. They concern some minor representation issues, the seed selection process, the search bias definition for hypothesis space description and the specification of the search strategy adopted by each task,. In this section, solutions adopted in the last release of the learning system ATRE are illustrated.

As to the representation issues, ATRE adopts the representation language of literals, where literals are in form of function symbols. The language of observation is object-centered and the language of hypothesis is that of linked[1], range-restricted[2] definite clauses.

---

[1] Literals that share at least a variable with another clause literal.

[2] Range restrictedness states that each variable that appears in the clause head has also to appear in the clause body.

Seed selection is a critical point. In the example of Fig.1, if the search had started from *even(2)* and *odd(1)*, the first clause added to the theory would have been *odd(X)* ← *succ(Y,X), zero(Y)*, thus resulting in a less compact, though still correct, theory for odd and even numbers. Therefore, it is important to explore the specialization hierarchies of several seeds for each predicate. When training examples and background knowledge are represented either as sets of ground atoms (flattened representation) or as ground clauses, the number of candidate seeds can be very high, so the choice should be stochastic. The object-centered representation adopted by ATRE has the advantage of reducing the number of candidate seeds by partitioning the whole set of training examples E into *training objects*. In particular, objects in ATRE are in form of ground multiple-head clauses where clause heads contain a subset of positive and negative examples; this provides a comprehensible and compact way to represent observations. The main assumption made in ATRE is that *each object contains examples explained by some base clauses of the underlying recursive theory*.[3] Therefore, by choosing as seeds *all* examples of different concepts represented in one training object, it is possible to induce some of the correct base clauses. Since in many learning problems the number of positive examples in an object is not very high, a parallel exploration of all candidate seeds is feasible. Mutually recursive concept definitions will be generated only after some base clauses have been added to the theory.

Seeds are chosen according to the textual order in which objects are input to ATRE. If a complete definition of the predicate p_j is not available yet at the i-th step of the separate-and-conquer search strategy, then there are still some uncovered positive examples of p_j. The first (seed) object O_k in the object list that contains uncovered examples of p_j is selected to generate seeds for p_j.

Generally, an important problem in ILP settings concerns the definition of the hypothesis space itself. However, in some cases, the user has a clear idea of relevant properties that should appear in the body of clauses. A *language bias* has been defined in ATRE to allow users to express constraints that should be satisfied by interesting clauses in the specialization hierarchy. In its current version, the language bias includes the following declarations:

$$starting\_number\_of\_literals(p_i,N)$$
$$starting\_clause(p_i,[L_1,L_2,...,L_N])$$

where p_i is a target predicate, N is a cardinal number, and $[L_1,L_2,...,L_N]$ represents a list of literals. In particular, the *starting_number_of_literals* declaration specifies the initial length of interesting clauses (at least N literals in the body), while the *starting_clause* declaration specifies a conjunctive constraint on the body of an interesting clause: all literals in the list $[L_1,L_2,...,L_N]$ must occur in the clause. Multiple *starting_clause* declarations for the same target predicate p_i specify alternative conjunctive constraints for interesting clauses of specialization hierarchies associated to p_i. In addition, the following declaration:

$$starting\_literal(p_i,[L_1,L_2,...,L_N])$$

[3] Problems caused by incomplete object descriptions violating the above assumption are not investigated in this work, since they require the application of *abductive* operators, which are not available in the current version of the system.

specifies a disjunctive constraint at literal level for the body of interesting clauses. Literals are expressed as follows:

$$f(\text{decl-arg}_1, \ldots, \text{decl-arg}_n) = \text{Value}$$
$$g(\text{decl-arg}_1, \ldots, \text{decl-arg}_n) \in \text{Range}$$

where *decl-arg*'s are mode declarations for predicate arguments. Declarations are applicable only to variables and influence the way of generating variables. Two modes are available in the current version of ATRE: *old* and *new*. The first mode means that the variable is an input variable, that is, it corresponds to a variable already occurring in the clause. The second mode means that the variable is a new one. Furthermore, values and ranges of predicates can be ground or not.

This search bias can be profitably exploited both to define a general basic structure of the theory and to restrict the number of hypotheses to evaluate during the search.

The last undefined point of the search strategy concerns the search strategy adopted by each task. ATRE applies a variant of the beam-search strategy that traverses each specialization hierarchy top-down. The system generates all candidate clauses at level *l+1* starting from those filtered at level *l* in the specialization hierarchy. During task synchronization, which occurs level-by-level, the best *m* clauses are selected from those generated by all tasks. The user specifies the beam of the search, that is *m*, and a set of preference criteria for the selection of the best *m* clauses.

2.5 Algorithmic details of the learning strategy

Variations on the separate-and-conquer strategy proposed in the previous sections have been implemented in ATRE. In Fig.2, the resulting algorithm is reported.

Starting with an empty theory, the algorithm implements a conquer stage that performs a general-to-specific beam search to generate a set of consistent clauses for the concepts to be learned. At each step of the search space exploration, for each concept to be learned, the root of each specialization hierarchy is generated, an object is selected and a number of seeds are generated. Then, while a search stopping criterion (e.g. a user-defined number of consistent clauses) does not succeed, for each seed, all rules in its associated hierarchy are specialized by adding a literal. By means of these two nested loops, the parallel mechanism illustrated in Section 2.2 is implemented. Consistent and range-restricted clauses are then selected from the set of specialized clauses. In addition, according to some user preference criteria, the set of most promising clauses are chosen.

When the search stopping criterion becomes true, the best generated clause is chosen according to some user preference criteria, the covered examples are separated from the set of training examples to cover and the logical theory is extended by adding the new best rule.

Finally, since the addition of a consistent clause may lead to an inconsistent theory, ATRE applies the layering technique presented in Section 2.1 (procedure *VerifyGlobalConsistency*) in order to restore the consistency.

```
procedure SEPARATE_AND_PARALLER_CONQUER(Concepts, Examples, PC)
  Theory = ∅
  Consistent = ∅
  Rules = ∅
  while (POSITIVE(Examples) ≠ ∅) OR (STOPCRITERIA(Theory, Examples))
    Seeds = ∅
    foreach Ci in Concepts do
      Rules = Rules∪{Ci ←[]}
      SeedObji = SELECT _OBJECT(Ci)
      Seeds = Seeds ∪ GENERATE_SEEDS(SeedObji)
    end foreach
    while (|Consistent| ≤ Minimum_number_of_consistent_rules)
      foreach S in Seeds do
        Ci = CONCEPT(S)
        Obji = OBJECT(S)
        foreach Rule_of_Ci in Rules do
          Specialized_rules = SPECIALIZE_RULES(Ci,Obji)
        end foreach
      end foreach
      Consistent = Consistent ∪ (CONSISTENT(Specialized_rules) ∩
                   RANGE_RESTRICTED(Specialized_rules))
      Rules = SELECT_BEST_RULES_TO_SPECIALIZE(Specialized_rules, PC)
    end while
    BestRule = SELECTBESTRULE(Consistent, Examples)
    Covered = COVER(BestRule,Examples)
    Examples = Examples \ Covered
    Theory = Theory ∪ BestRule
    VERIFYGLOBALCONSISTENCY(Theory)
    Concepts = CONCEPTS_TO_BE_LEARNED(Examples)
  end while
return(Theory)
```

Fig. 2. The separate-and-parallel-conquer algorithm. Variants of the separate-and-conquer algorithm are in bold. In the parallel foreach statement variables are assigned a value in parallel.

3. Application to document understanding

In this section, we present a real world application where the learning of dependencies among concepts is suggested by the intrinsic nature of the domain. To test ATRE in this domain, it has been integrated in WISDOM++

(http://www.di.uniba.it/~malerba/wisdom++), an intelligent document processing system that uses logical theories learned by ATRE to perform document image understanding [8]. In particular, ATRE can be trained to learn definitions of concepts of interest from examples provided by the user.

Generally speaking, a document is characterized by the layout structure, which aims to describe its internal organization, and by the logical structure, which describes the content of a document. More precisely, the former associates the content of a document with a hierarchy of layout components, while the latter associates the content of a document with a hierarchy of logical components. The leaves of the logical structure are the basic logical components, such as authors of a paper. The heading of an article, encompassing the title and the author, is therefore a composite logical component. The root of the logical structure is the document class. Here, the term document understanding denotes the process of mapping the layout structure of a document into the corresponding logical structure. The document understanding process is based on the assumption that documents can be understood by means of their layout structures alone. In WISDOM++ it consists in the association of the most abstract level of the layout hierarchy with basic logical components. The task can be performed by means of a set of rules which can be generated automatically by learning from a set of training objects. Each training object describes the layout of a document image and the logical components associated to layout components (see Fig. 3).

The following is an example of the body of a training observation automatically generated by WISDOM++:

```
part_of(1,2)=true,…,part_of(1,13)=true,
width(2)=391,…,width(13)=263,
height(2)=9,…,height(13)=58,
type_of(2)=text,…,type_of(13)=text,
x_pos_centre(2)=354,…,x_pos_centre(13)=411,
y_pos_centre(2)=29,…,y_pos_centre(13)=753,
on_top(2,4)=true,…,on_top(12,13)=true,
to_right(11,12)=true,…,to_right(3,6)=true,
alignment(3,8)=only_left_col,…,alignment(8,10)=only_upper_row
```

where the constant 1 denotes the whole page and the remaining constants 2,…,13 denote distinct layout components.

Conversely, in the head of a training observation the class name of the document, which unambiguously identifies the appropriate set of logical components, and the association of logical components with layout blocks is described as follows:

```
class(1)=tpami, affiliation(2)=false, …, paragraph(2)=false,
…
affiliation(15)=false,     page_number(15)=false,     figure(15)=false,
caption(15)=false,     index_term(15)=false,     running_head(15)=false,
author(15)=false,          title(15)=false,          abstract(15)=false,
table(15)=false,     formulae(15)=false,     subsection_title(15)=false,
section_title(15)=false,   biography(15)=false,   references(15)=false,
paragraph(15)=true
```

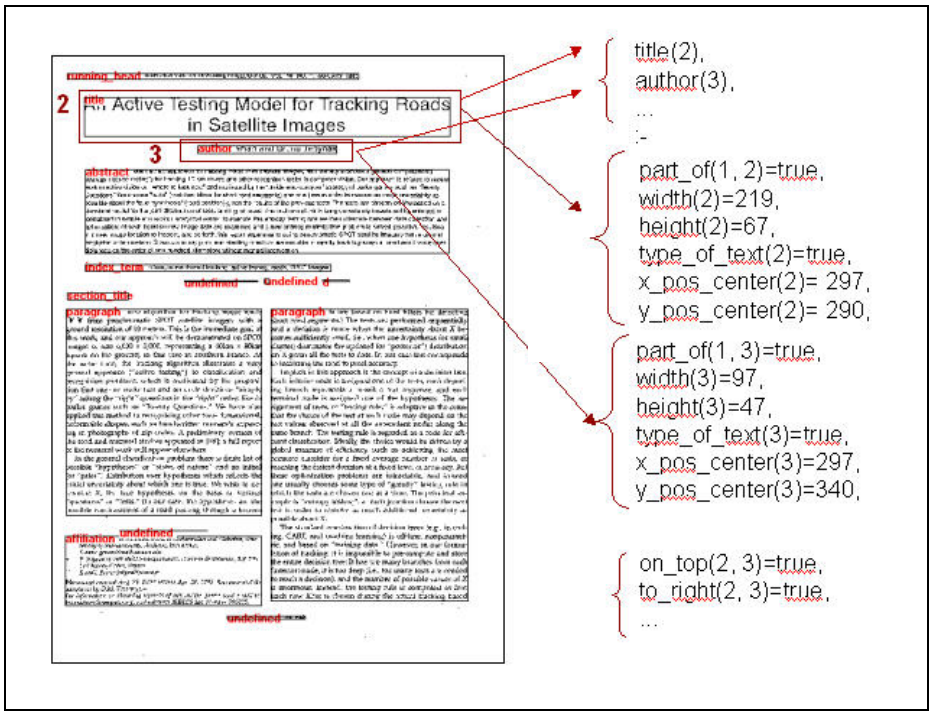

Fig. 3. Training description for document understanding.

By running ATRE on a document understanding dataset obtained from scientific papers, a set of theories is learned. Some examples of learned clauses follow:

```
author(X1) ← alignment(X1,X2)=only_middle_col, abstract(X2),
             height(X1)∈[7..13]
figure(X1) ← type_of(X1)= image, width(X1)∈[12..227],
             x_pos_centre(X1)∈[335..570]
references(X1) ← to_right(X1,X2), biography(X2),
                 width(X2)∈[261..265]
```

They can be easily interpreted. For instance, the first clause states that if a quite short layout component (X1), whose height is between 7 and 13, is centrally aligned with another layout component (X2) labelled as the abstract of the scientific paper, then it can be classified as the author of the paper. These clauses show that ATRE can automatically discover meaningful dependencies between target predicates but as we expect, it does not learn mutual dependencies since the domain seems to be not suitable for mutual definition learning.

In order to evaluate the predictive accuracy of theories induced by ATRE, a 5-fold cross-validation test has been performed. In particular, a dataset of 21 documents have been divided in five folds removing some documents in turn (Table 1).

For each fold, ATRE is trained on the remaining folds and tested on the hold-out fold. The number of omission/commission errors is recorded and showed in Table 2.

Fold number	List of documents	Number of pages	Number of positive examples
1	Tpami1	13	27
	Tpami13	3	12
	Tpami14	10	26
	Tpami 16	14	34
2	Tpami8	5	16
	Tpami15	15	35
	Tpami18	10	24
	Tpami24	6	16
3	Tpami3	15	34
	Tpami7	6	16
	Tpami12	6	14
	Tpami20	14	34
4	Tpami9	5	15
	Tpami11	6	15
	Tpami19	20	45
	Tpami21	11	25
5	Tpami4	14	31
	Tpami6	1	6
	Tpami10	3	11
	Tpami17	13	31
	Tpami23	7	18

Table 1. Distribution of pages and examples per document grouped by 5 folds.

Fold	Omission errors	Commission errors
1	21 / 99	15 / 4080
2	13 / 91	16 / 4697
3	17 / 98	15 / 4781
4	19 / 100	14 / 5318
5	31 / 97	8 / 3746
Average %	20,76	0,30
Std Dev. %	6,75	0,06

Table 2. Accuracy of induced theories in the first page documents task.

O*mission* errors occur when logical labelling of layout components are missed, while *commission* errors occur when wrong logical labelling are "recommended" by a rule. Since the former kind of error consists in a missing labelling operation while the latter leads to a wrong label assignment, it is straightforward that commissions are more critical errors than omissions.

The previous table shows that ATRE performs more omission errors (about 21%) than commission errors (about 0.30%).

4. Conclusions

In this paper, variations of the classical separate-and-conquer strategy have been proposed as means to overcome issues raised by the complex task of recursive theory

learning. Proposed solutions have been implemented in ATRE and tested in a real-world domain. As future work, improvements concerning the quality of induced theories are worth to be investigated.

References

1. Boström, H.: Induction of Recursive Transfer Rules. In J. Cussens (ed.), Proceedings of the Language Logic and Learning Workshop (1999) 52-62.
2. Buntine, W.: Generalised subsumption and its applications to induction and redundancy. Artificial Intelligence, Vol. 36 (1988) 149-176.
3. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about Datalog (and never dared to ask). IEEE Trans. on Knowledge and Data Engineering 1(1) (1989) 146-166.
4. De Raedt, L., Dehaspe, L.: Clausal discovery. Machine Learning Journal, 26(2/3) (1997) 99-146.
5. De Raedt, L., Lavrac, N.: Multiple predicate learning in two Inductive Logic Programming settings. Journal on Pure and Applied Logic, 4(2) (1996) 227-254.
6. Khardon, R.: Learning to take Actions. Machine Learning, 35(1) (1999) 57-90.
7. Malerba, D.: Learning Recursive Theories in the Normal ILP Setting, Fundamenta Informaticae, 57(1) (2003) 39-77.
8. Malerba, D., Esposito, F., Lisi, F.A., Altamura, O. (2001). Automated Discovery of Dependencies Between Logical Components in Document Image Understanding. *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, Seattle (WA), pp. 174-178.
9. Muggleton, S., Bryant, C.H.: Theory completion using inverse entailment. In: J. Cussens and A. Frisch (eds.): Inductive Logic Programming, Proceedings of the 10th International Conference ILP 2000, LNAI 1866, Springer, Berlin, Germany (2000) 130-146.
10. Nienhuys-Cheng, S.-W., de Wolf, R.: The Subsumption theorem in inductive logic programming: Facts and fallacies. In: De Raedt, L. (ed.): Advances in Inductive Logic Programming. IOS Press, Amsterdam (1996) 265-276.
11. Plotkin, G.D.: A note on inductive generalization. In: Meltzer, B., Michie, D. (eds.): Machine Intelligence 5. Edinburgh University Press, Edinburgh (1970) 153-163.
12. Plotkin, G.D.: A further note on inductive generalization. In: Meltzer, B., Michie, D. (eds.): Machine Intelligence 6. Edinburgh University Press, Edinburgh (1971) 101-124.
13. Rouveirol, C.: Flattening and saturation: Two representation changes for generalization. Machine Learning Journal, 14(2) (1994) 219-232.