Knowl Inf Syst (2007) 11(2): 217–242
DOI 10.1007/s10115-006-0019-5

Knowledge and Information Systems

REGULAR PAPER

F. Esposito · S. Ferilli · T. M. A. Basile · N. Di Mauro

Inference of abduction theories for handling incompleteness in first-order learning

Received: 9 May 2005 / Revised: 1 November 2005 / Accepted: 14 January 2006 /
Published online: 23 March 2006

Abstract In real-life domains, learning systems often have to deal with various kinds of imperfections in data such as noise, incompleteness and inexactness. This problem seriously affects the knowledge discovery process, specifically in the case of traditional Machine Learning approaches that exploit simple or constrained knowledge representations and are based on single inference mechanisms. Indeed, this limits their capability of discovering fundamental knowledge in those situations. In order to broaden the investigation and the applicability of machine learning schemes in such particular situations, it is necessary to move on to more expressive representations which require more complex inference mechanisms. However, the applicability of such new and complex inference mechanisms, such as abductive reasoning, strongly relies on a deep background knowledge about the specific application domain. This work aims at automatically discovering the meta-knowledge needed to abduction inference strategy to complete the incoming information in order to handle cases of missing knowledge.

Keywords Incomplete knowledge · Inductive Logic Programming · Abduction

1 Introduction

Various kinds of imperfections in data such as noise, incompleteness and inexactness, typical of real-life domains, often affect the learning systems effectiveness. Specifically, noise takes the form of random errors in both the training examples and the background knowledge; incomplete data consist of too sparse training examples from which it is difficult to reliably detect correlations; inexactness refers to the description language being inappropriate because it does not allow/facilitate an exact representation of the target concept. Usually learning systems exploit a

F. Esposito (✉) · S. Ferilli · T. M. A. Basile · N. Di Mauro
Department of Computer Science, University of Bari, Bari, Italy
E-mail: {esposito, ferilli, basile, ndm}@di.uniba.it

single mechanism, often called noise-handling mechanism, for dealing with such kinds of imperfect data. Even more difficult to deal with are missing values, that are usually handled by a separate mechanism. Many noise-handling mechanisms have been proposed while not many learning systems are able to handle missing data, thus a knowledge engineer is usually called to solve this problem before running a learning system. Furthermore, most of the learners dealing with incomplete information are attribute-value learners that are unable to handle relational domains. To this purpose Inductive Logic Programming (ILP) systems were developed [22] that are based on a first-order logic representation. A further step in dealing with relational domains is represented by the work of [15] in which the authors present an induction technique that discovers classification rules from examples using second-order relations as a representational model. First-order logic is an expressive representation but it is computationally expensive, so it is natural to consider improving the performance of inductive logic data mining. In [28] the authors proposed the exploitation of a parallelization technique for inductive logic, and implemented a parallel version of a core inductive logic programming system. However, no noise handling mechanism was proposed for such a parallelization.

The methods proposed in literature to face the missing values problem act in different ways. The most frequent is to replace a missing value in an example by the majority value of the attribute/argument within the class the example belongs to (LINUS [22]). Others exploit a Kernel Density Estimation-based algorithm for clustering in large multimedia databases to overcome the limits of the classical clustering algorithms in dealing with the large amount of noise (DENCLUE [16]). Another one is to replace an example having a missing value with several examples, one for each of the possible attribute values, weighted by the conditional (with regard to the class of the example) probabilities of the values (ASSISTANT [2], CN2 [4], LINUS). The last one is to replace an example having a missing value with many examples, each with one possible value of the attribute type (LINUS).

In general, these strategies adopt different methods to first complete the missing information (in some case with the additional support of a knowledge engineer) and then learn from the completed data. Two problems can arise with them. First, it is difficult to compute statistics and probabilities to fill in the missing values, as for domains in which new examples could be available during the learning process thus requiring an incremental capability of the systems. The second issue concerns both the impossibility to know a priori the whole set of descriptors that make up the representation language and the limiting implicit assumption that a knowledge engineer must monitor the pre-processing aimed at completing the examples.

These observations led to design strategies that would dynamically handle incomplete information within the learning process. An in-depth analysis of this landscape revealed that the limitation of most traditional Machine Learning approaches is due to the fact that they exploit simple or constrained knowledge representations for the sake of efficiency and are based on single inference [23]. This suggested the exploitation of purposely designed mechanisms in order to broaden the investigation and the applicability of machine learning schemes. Specifically, it is necessary to move on to more expressive representations, which in turn require more complex inference mechanisms.

A general schema for the concept-learning paradigm is provided by the *fundamental equation for inference* [23]: $BK \cup T \models O$ that involves a language $\mathcal{L}$, for which in this work the *single representation trick* [5] will be assumed, a background knowledge BK and a theory T, that contains concept definitions accounting for some observations O. Specifically, O stands for the extensional representation of concepts, while T is an intensional description, expressed in $\mathcal{L}$, that explains such concepts together with BK. Deduction "traces forward" the equation, deriving O given T and BK, and hence it is a truth-preserving inference. Conversely, tracing the equation "backward" yields two falsity-preserving inferences (meaning that if O is false, then the hypothesis cannot be true): *induction*, when T is to be hypothesized given O and BK, or *abduction*, when BK is to be hypothesized given O and T (i.e., plausible/likely causes of given observations).

Most traditional approaches to concept-learning rely on inductive mechanisms to fine-tune T in order to achieve the learning goal, but problems might arise due to the partial relevance of the available evidence O. Abduction could be exploited to overcome such a limitation by bridging the observations' relevance gap. Indeed, it is able to capture *default reasoning* [26], a well-known form of reasoning to deal with incomplete information [19, 24]. Thus, making these inference strategies work together would allow to take advantage of the benefits that each of them can bring. Many studies presented in the literature aimed at enforcing such an integration within an ILP framework in a principled way, dealing with incomplete information based on an underlying theory of abduction so to combine in a nontrivial fashion ILP learning methods and abduction methods. A step in this direction is proposed in [20], where the Authors show how it is possible to learn with incomplete background information about the training examples by exploiting the hypothetical reasoning of abduction. Specifically, the deductive proof procedure of logic programming is replaced by an abductive proof procedure for Abductive Logic Programming [19] (see Sect. 2). Furthermore, in [9, 12, 21] the Authors proposed and developed a framework for the integration of abductive and inductive learning in an ILP system able to incrementally perform the learning task.

However, the research and literature so far assumed that the information needed to the learning systems to exploit the hypothetical reasoning of abduction in support of induction within the learning task is provided by a knowledge engineer. The objective in this work is the automatic inference of such information. To this aim we developed a procedure that, starting from the training data, generates a set of special rules to be exploited in the abductive proof procedure supporting the standard inductive reasoning.

This paper is organized as follows. Next section describes the general framework integrating inductive and abductive inference mechanisms to handle the imperfect data situation. Then, the techniques for automatically inferring the meta-knowledge needed to carry out abductive reasoning are proposed. Finally, some experiments are reported showing the effectiveness of the proposed methods in both artificial and real-world datasets.

2 Abduction for handling missing values: the general framework

As a mechanism for knowledge assimilation, abduction can be employed when observations about the world are given and must be assimilated into a knowledge base [7]. As already pointed out, from an inferential point of view abduction and induction are similar since both are falsity-preserving. However, abduction is generally understood as reasoning from effects to causes (or explanations), while induction concerns the inference of general rules from specific data. Abduction requires an initial theory containing the conditions that can be involved in the construction of the explanation. These can be made explicit by means of abductive inferences, and subsequently exploited by inductive mechanisms to synthesize new knowledge, that in turn can be exploited by subsequent abductive inference to build new explanations.

The general schema of an inductive learning algorithm [6] can be extended with an abductive proof procedure. Hence, the problem of abduction can be formalized as follows [7]: **Given** a theory T, including also the background knowledge, some observations O and some constraints I, **Find** an explanation H such that $T \cup H$ is consistent, $T \cup H$ satisfies I and $T \cup H \models O$.

Candidate abductive explanations H should be described in terms of domain-specific predicates, referred to as *abducibles*, that are not (completely) defined in T, but may contribute to the definition of other predicates. They carry all the incompleteness of T: if it is possible to complete these predicates then the theory would be correctly described. The integrity constraints I should provide indirect information about these abducible predicates [19].

2.1 Abductive logic programming

Abductive Logic Programming (ALP) [10, 21] is an extension of Logic Programming to support abductive reasoning with theories (logic programs) that incompletely describe their problem domain. In ALP this incomplete knowledge is captured by an abductive theory, defined as a triple made up by a (hierarchical) logic program T, a set of abducible predicates $\mathcal{A}$, and a set of integrity constraints $\mathcal{I}$ represented as program clauses.

An abductive procedure can be exploited to deal with the problem of incompleteness by finding explanations that make hypotheses (abductive assumptions) on the state of the world, possibly involving new abducible concepts. The procedure is generally goal-driven by the observations that it tries to explain. Preliminarily, the top-level goal undergoes a transformation process that converts it into sub-goals. This provides a simple and unique modality for dealing with non-monotonic reasoning. Algorithm 2.1 sketches the classical abductive proof procedure proposed in [17]. After a literal is selected, if it is not abducible or a default one (A1), the procedure continues with a resolution step with clauses from T. Otherwise, if the fact has been already assumed abductively (and consistently) as true in previous steps (A2) it can be dropped (a case of successful proof). Otherwise (A3), a new fact may be assumed as true, provided that it is consistent with the current integrity constraints $\mathcal{I}$, which is verified by the consistency-check subroutine reported in Algorithm 2.2.

Algorithm 2.1 Abductive Refutation Algorithm
abduce(T, G, Δ, $\mathcal{AD}$, $\mathcal{I}$)

{**input:** T: theory, G: Datalog goal (set of literals), Δ: initial abductive assumptions, $\mathcal{AD}$: the set of abducibles and default literals, $\mathcal{I}$: the integrity constraints;
output: Δ' final abductive assumptions;}
$\Delta' = \Delta$;
while $G \neq \emptyset$ **do**
  $L :=$ Select a literal from G;
  if $L \notin \mathcal{AD}$ **then**
    /* (**A1**) */ $G :=$ Resolvent of some clause of T with G on L;
  else if $L \in \Delta'$ **then**
    /* (**A2**) */ $G := G \setminus L$;
  else if $\overline{L}_J \notin \Delta'$ and $\exists \Delta_C = \mathit{consistency}(T, L, \Delta' \cup \{L\}, \mathcal{AD}, \mathcal{I})$ **then**
    /* (**A3**) */ $G := G \setminus L$; $\Delta' := \Delta_C$;

Algorithm 2.2 Consistency Derivation Algorithm
consistency(T, L, Δ, $\mathcal{AD}$, $\mathcal{I}$)

{**input:** T: theory, $L \in \mathcal{AD}$: a literal, Δ: initial abductive assumptions,
$\mathcal{AD}$: the set of abducibles and default literals, $\mathcal{I}$: the integrity constraints;
output: Δ' final abductive assumptions;}
$\Delta' := \Delta$;
$C := \bigcup$ of goals of the form : $-L_1, L_2, \ldots, L_n$ obtained by resolving the abducibles or default literal L with the integrity constraints $\mathcal{I}$ with no such goal been empty;
while $C \neq \emptyset$ **do**
  $B :=$ Select a goal from C; $M :=$ Select a literal from B;
  if $M \notin \mathcal{AD}$ **then**
    $H :=$ Resolvent of some clause of T with B on M;
    $C := \{C \setminus B\} \cup H$;
  else if $M \in \mathcal{AD}$ and $M \in \Delta'$ **then**
    /* (**F1**) */ $H := B \setminus M$; $C := \{C \setminus B\} \cup H$;
  else if $M \in \mathcal{AD}$ and $\overline{M} \in \Delta'$ **then**
    /* (**F2**) */ $C := C \setminus B$;
  else if $M \in \mathcal{AD}$ and ($M \notin \Delta'$, $\overline{M} \notin \Delta'$) **then**
    /* (**F3**) */
    if $\exists \Delta_A = \mathit{abduce}(T, \overline{M}, \Delta', \mathcal{AD}, \mathcal{I})$ **then**
      $C := C \setminus B$; $\Delta' := \Delta_A$

The various branches in the consistency-check subroutine are similar to derivations except that, when dealing with an abducible or a default literal, if it has already been abduced (F1) then it is simply dropped (i.e., consistency is trivially proved); otherwise, if its complement has already been abduced or can be abduced (F2), the entire goal is dropped. In the last if-branch (F3), whenever the literal to be

tested is an abducible or default one, but neither it nor its complement have been already abduced, the abductive procedure is called, in order to try hypothesizing it by abduction. Thus, the two procedures may call each other both when a new abductive assumption requires further consistency checks against the constraints and vice-versa.

Representing theories as *hierarchical* logic programs allows to maintain the Least Herbrand Models semantics, coping with negation by means of NAF [3] rule. Indeed, since the language of definite clauses with integrity constraints has been proven to subsume NAF [8], integrity constraints can be simulated using NAF as well. The advantage of adopting this semantics resides in the fact that $T \models P_1,\ T \models P_2,\ \ldots,\ T \models P_n$ implies that $T \models P_1 \wedge P_2 \wedge \cdots \wedge P_n$. Hence, positive/negative examples can be tested separately for completeness/consistency, that is fundamental in a theory revision context, since in the incremental learning process one cannot assume to have all the examples available at any time.

2.2 Extending an inductive learning framework with an abductive proof procedure

Algorithm 2.3 sketches the integration of an incremental inductive learning framework with an abductive proof procedure as proposed in [12]. Here, M represents the set of all positive and negative processed examples, E is the example currently examined, T is the theory generated so far according to M, $AbdT$ is the abduction theory, D is the set of facts hypothesized by the abductive derivation when successfully applied to a goal in theory T. *Generalize* and *Specialize* are the inductive operators used by the system to refine an incorrect theory.

Algorithm 2.3 Theory Revision extending an incremental inductive learning framework with an abductive proof procedure

Revise (T: theory, E: example, M: historical memory, $AbdT$: Abductive Theory);
$D \leftarrow E$
if ($Abductions$ = Abduce(T, E, D, $AbdT$)) succeeds **then**
  Add to D the abduced literals $Abductions$; $M \leftarrow M \cup \{E \cup D\}$;
else $M \leftarrow M \cup E$
  if E is a positive example **then** Generalize(T, E, M);
  else Specialize(T, E, M);

The incremental system works by checking on each new example whether the currently learned theory is able to correctly classify it. If an (omission/commision) error occurs, before performing a revision of the theory, the system checks whether the example can be correctly explained by hypothesizing new facts by means of the abductive procedure reported in Algorithm 2.1. Only in case of failure the refinement operators are fired. Abduction is thus exploited to complete the observations in such a way that the corresponding examples are either covered (if positive) or ruled out (if negative) by the already generated theory, thus avoiding, whenever possible, the use of the operators to modify/revise the theory. The set of abduced literals for each observation is minimal, which ensures that abducibles are used only when really needed. Since specific facts are not allowed in the learned

theory, the abduced information is attached directly to the observation that generated it, so that the "completed" examples obtained this way will be available for subsequent refinements of the theory. Such information will also be available to subsequent abductions, in order for them to preserve consistency among the whole set of abduced facts. To sum up, when a new observation is available, the abductive proof procedure is started, parameterized on the current theory, the example and the current set of past abductive assumptions. If the procedure succeeds, the resulting set of assumptions, that were necessary to correctly classify the observation, is added to the example description before storing it, otherwise the usual refinement procedure (generalization/specialization) is performed.

3 Learning meta-knowledge for abductive inference

The abductive proof procedure presented in Sect. 2 requires that an abductive theory for the specific application domain is available. In the current practice, it is in charge of the human expert to specify it. Of course quality, correctness and completeness in the formalization of such meta-information can affect the feasibility of the learning process. Providing it is a very difficult task, also, that requires a deep knowledge of the application domain, and is in any case an error-prone activity, since omission errors may take place for a number of reasons. For instance, the domain and/or the language used to represent it might be unknown to the experimenter, because datasets are provided by third parties. In any case, it is not easy for non-experts to single out and formally express such parameters, just because they are not familiar with the representation language needed by the automatic systems and the related technical issues. Other possible causes include the large number of parameters to be specified, and the fact that they are sometimes hidden from the normal focus-of-attention.

These considerations would make it highly desirable to develop procedures that can automatically generate such information starting from the same observations that are input to the learning process in order to make the learning system completely autonomous. Hence, a strong motivation for the research presented in this paper, aimed at proposing algorithms to automatically infer the meta-information required to carry out abductive inference. The challenge in this attempt is that it does not try to learn something *about* the given instances, but instead aims at gathering information on the domain and/or its description *from* the given instances. This means that we are no more concerned with the description of concepts by proper juxtaposition of literals, but rather with the meaning underlying the language used. Thus, the problem deals with semantics rather than with syntax.

3.1 Abducibles and integrity constraints

In setting up an abductive logic programming task, the logic program is typically to be learnt, while abducibles and integrity constraints have to be provided by the domain expert. Thus, a first problem is deciding on which properties and/or relations abduction can be carried out, i.e., listing the abducibles. Indeed, abductive

reasoning needs to know on which concepts abductions (i.e., guesses about unknown facts) can be made. In the absence of further information, it is possible to assume that any hypothesis that can help in solving the problem at hand is useful, and the automatic system should be allowed to carry it out. Thus, a straightforward solution could be including in the set of abducibles all predicates that make up the description language, in order to provide the abductive reasoner with all the freedom it needs for hypothesizing information. In fact, if an intensional background knowledge is present, some of such predicates might have a definition, which contradicts one of the requirements for abducibles. Thus, a better solution is taking into account only the subset of "basic" predicates in the description language that occur in the available observations, for which it holds the requirement of not having a definition in the theory.

The other issue, far more complex, concerns the definition of the integrity constraints. It is, at the same time, a fundamental and difficult task, whose quality can determine the very feasibility of the learning process. Hence, the need for a research on the possibility of automatically learning such constraints, this way overcoming possible problems related to omissions and/or wrong formalization on the part of the human expert.

Learning denials (the form in which integrity constraints are coded in an abductive theory) cannot be simply cast as a supervised learning task, since it aims at inducing rules whose head is empty. Rather, it can be seen as a specific case of unsupervised learning aimed at finding regularities (specifically, conditions that are never verified) in a first-order logic database. Thus, the data mining approaches are better suited to carry out this task. Some systems are present in the literature that can learn denials. One of them, often referred to, is Claudien [25], that actually implements a more general algorithm for finding regularities that occur in a set of unlabelled observations represented as facts. It requires a template of the clauses to be induced, and can limit the corresponding search space using heuristics and resource bounds. By properly setting its parameters, it can be applied for learning classification rules, association rules, clauses and also denials. Such a system inspired a number of successive works, among which the development of the systems Primus and its successor Tertius [14]. They are based on the generation of possible (H, B) couples, where H and B are sets of literals in the given description language to be interpreted, possibly negated, as candidate head and body, respectively, of a clause to be generated. The frequency with which each candidate rule is (or is not) verified in the dataset is computed, and statistical approaches are exploited to decide if such frequencies are significant, in which case a corresponding rule is generated. Background knowledge (i.e., derived predicates such as *ancestor* in a family environment) can be used, but increasing the number of literals in H and B causes high computational costs, thus sampling and non-redundant operators are exploited. Aleph[1] is another widely known learning system to induce integrity constraints, but no reference is available for this specific feature except the user manual statement that it works in a similar way as Claudien. All of these systems can actually learn denials, but this is just a specific setting or a side-effect of a wider range of possibilities that the implemented algorithms provide. Thus, the aim of this paper is devising simpler procedures, purposely devoted to the

[1] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/

generation of integrity constraints for an abductive theory, that being limited to this specific task can carry out it in a more focused and effective way.

The starting point in doing this is the fact that integrity constraints represent situations that cannot occur in the described world. Thus, the available observations cannot actively help in defining them. Rather, the aim is identifying combinations of descriptors and of the related arguments that cannot hold. In doing so, one possible strategy is generating a number of such combinations, according to a given strategy, and then exploiting the available observations passively to check if the generated combination occurs in at least one case or not. In the former case, it cannot be a constraint, according to the assumption that observations are correct and report only true information. In the latter case, this can be taken as a suggestion, but not as a guarantee (since its absence could be due to just the fact that by chance that situation did not ever occur in the specific observations at hand), that the combination does not occur because it in fact makes no sense in the considered world. This, of course, raises the problem of having a set of observations that is significant not only numerically, but also in the sense that they depict a significant amount of different cases.

Now, the next step is about how to generate the literals (and variables) combinations to be tested: Generating and testing all possible combinations becomes soon impossible even for relatively small datasets; Bounding the cardinality of the combinations to be generated to a given l, although useful, is not sufficient to avoid the combinatorial explosion. Thus, it is necessary to identify specific classes of constraints that can be considered meaningful in general (i.e., dataset independent) and thus are worth checking. A first important class is that of object properties, represented by unary predicates. Indeed, it is undoubtedly interesting to know which combinations of attributes are (im-)possible for a given object, in order for the abductive proof procedure to avoid them (e.g., it generally holds that a line is either tall or wide, but cannot be both at the same time). In this case, the problem can be significantly simplified since the presence of just one variable in the predicates allows to focus on just the predicates combinations, excluding the generation of duplicate literals and the presence of unrelated variables. Algorithm 3.1 sketches the procedure.

Algorithm 3.1 Induction of Integrity Constraints made up of unary predicates

Create_Constraints(N; $\mathcal{E}$; *UnaryPreds*; *NotConstraints*; *Constraints*);
{**input:** N: Maximal cardinality of constraints to generate; $\mathcal{E}$: Set of observations; *UnaryPreds*: Set of Unary Predicates;
output: *NotConstraints*: Set of non-Constraints; *Constraints*: Set of Integrity Constraints;}
$NotConstraints := \emptyset$; $Constraints := \emptyset$;
for all $a, b \in UnaryPreds, a \neq b$ **do**
  if $\mathcal{E} \vdash \{a(X), b(X)\}$ **then**
  $NotConstraints := NotConstraints \cup \{\{a(X), b(X)\}\}$
  else $Constraints := Constraints \cup \{\{a(X), b(X)\}\}$
for $n := 3..N$ **do**
  for all $NC \in NotConstraints, |NC| = n - 1$ **do**
    for all $a(X) \in UnaryPreds$ **do**
      if $\mathcal{E} \vdash \{\{a(X)\} \cup NC\}$ **then**
        $NotConstraints := NotConstraints \cup \{\{a(X)\} \cup NC\}$

else
if *not_trivial*(*Constraints*, $\{a(X)\} \cup NC$) **then**
Constraints := *Constraints* $\cup$ $\{\{a(X)\} \cup NC\}$

NotConstraints and *Constraints* are the lists of the currently identified non-constraints and constraints, respectively. Once a potential constraint is built, if it does not occur in the observations it is added to the list of constraints, provided that it is not a superset of some other (shorter) constraint (*not_trivial* function). In the first step, all possible n-tuples (with $2 \leq n \leq N$ for a fixed N) of unary predicates, all applied to the same variable, are generated and checked for occurrence in the available observations. First, all pairs of unary predicates are generated and checked for occurrence: those that are not satisfied by the observations are considered constraints and added to the *Constraints* list; conversely, those that happen at least once are added to the *NotConstraints* list. Then, all non-constraints of cardinality 2 are extracted from *NotConstraints* and extended with one more unary predicate, checked for occurrence and added to *NotConstraints* or *Constraints* accordingly. Then, all newly found non-constraints of cardinality 3 are extended and checked, and so on until the fixed N is reached.

However, although very useful, constraints on properties are not sufficient. It is often important, for the purpose of learning a significant abduction theory, to consider also constraints built on n-ary predicates. Without loss of generality, in this work we restrict to binary predicates, and propose a set of typical relationships among the arguments that appear in pairs of such predicates that are deemed as significant to be exploited as constraints. Specifically, the combinations that we propose to check are as follows.

Definition 3.1 (Binary Predicate Properties) Let be p_1 and p_2 be two (not necessarily distinct) binary predicates of the representation language, and X, Y and Z be three variables. We define the following properties:

- reflexivity: $\{p_1(X, X)\}$ (or $\{p_2(X, X)\}$);
- symmetry: $\{p_1(X, Y), p_2(Y, X)\}$;
- transitivity: $\{p_1(X, Y), p_2(Y, Z)\}$;
- convergence: $\{p_1(X, Y), p_2(Z, Y)\}$;
- divergence: $\{p_1(X, Y), p_2(X, Z)\}$.

In the next step, all binary predicates are considered, and checked for occurrence of the reflexive, symmetric, transitive, converging and diverging relationships. Again, when a relationship has no counterpart in the available observations, it is added to the *Constraints*, otherwise it is added to the *NotConstraints*. Lastly, all possible combinations of non-constraints on binary predicates and on unary predicates (applied to any of the variables appearing in the former), whose cardinality does not exceed the fixed N, are checked for occurrence and added to the *Constraints*, if it is the case, according to Algorithm 3.2.

It starts the process taking as input the list of *non-constraints*, both unary and binary, built so far. *UnaryNotConstrs* and *BinaryNotConstrs* are the sets of non-constraints found in the previous steps. Since all constraints on unary predicates have at least cardinality 2, a preliminary step in which all possible combinations of constraints on binary predicates with a single unary predicate must be separately

checked. Note that, in this step, no candidate constraint can be trivial, since its binary component is not a constraint by itself and its unary component is just a singleton. Conversely, in the loop that arranges unary and binary constraints, the only way a constraint can be trivial is being a superset of a constraint obtained in the previous loop, since none of its components is a constraint by itself.

Algorithm 3.2 Induction of Integrity Constraints made up of both unary and binary predicates

Create_constraints_with_binary_unary_literals(*N*; *Unary*; *Constraints*; *UnaryNotConstrs*; *BinaryNotConstrs*);
{**input:** *N*: Maximal cardinality of constraints to generate; $\mathcal{E}$: Set of observations; *Unary*: Set of Unary Predicates; *UnaryNotConstrs*: Set of non-constraints made up of unary predicates; *BinaryNotConstrs*: Set of non-constraints made up of binary predicates;
output: *Constraints*: Set of Integrity Constraints made up of unary or binary predicates;}
for all $NC \in BinaryNotConstrs$, $X \in vars(NC)$, $p \in Unary$ **do**
  if $|NC| < N \wedge \mathcal{E} \nvdash NC \cup \{p(X)\}$ **then**
    $Constraints := Constraints \cup \{\{p(X)\} \cup NC\}$
for all $BNC \in BinaryNotConstrs$ **do**
  V := vars(BNC); *TempConstraint* := BNC;
  for all $S \subseteq V$ **do**
    Apply a $UNC \in BinaryNotConstrs$ to each $X \in S$;
    Add it to *TempConstraint;*
    if $|TempConstraint| \leq N \wedge \mathcal{E} \nvdash TempConstraint$ **then**
      if *not_trivial*(*Constraints*, *TempConstraint*) **then**
        $Constraints := Constraints \cup \{TempConstraint\}$

Example 3.1 Consider the description language made up of the predicates: {block/1, line/1, low/1, medium/1, high/1, narrow/1, wide/1, part_of/2, on_top/2, to_right/2}. Let the available observations be: {part_of(a,b), part_of(a,c), part_of(a,d), part_of(a,e), part_of(a,f), line(b), medium(b), narrow(b), block(c), high(c), wide(c), line(d), low(d), wide(d), block(e), medium(e), wide(e), block(f), medium(f), wide(f), on_top(d,b), on_top(d,e), on_top(d,f), on_top(b,c), on_top(e,c), on_top(f,c),to_right(b,e), to_right(f,b)} (representing the block world in Fig. 1) and N be fixed to 4.

– *Step 1*:
  – Pairs of unary predicates:
    Constraints $= \{\{block(X), line(X)\},$
    $\{block(X), low(X)\}, \{block(X), narrow(X)\}, \{line(X), high(X)\},$
    $\{low(X), medium(X)\}, \{low(X), high(X)\}, \{low(X), narrow(X)\},$
    $\{medium(X), high(X)\}, \{high(X), narrow(X)\}, \{narrow(X), wide(X)\}\}$
    NotConstraints $= \{\{block(X), medium(X)\}, \{block(X), high(X)\},$
    $\{block(X), wide(X)\}, \{line(X), low(X)\}, \{line(X), medium(X)\},$
    $\{line(X), narrow(X)\}, \{line(X), wide(X)\}, \{low(X), wide(X)\},$
    $\{medium(X), narrow(X)\}, \{medium(X), wide(X)\}, \{high(X), wide(X)\}\}$
  – Triplets of unary predicates (extending couples of *NotConstraints*):
    Constraints $= \{\{block(X), medium(X), narrow(X)\},$

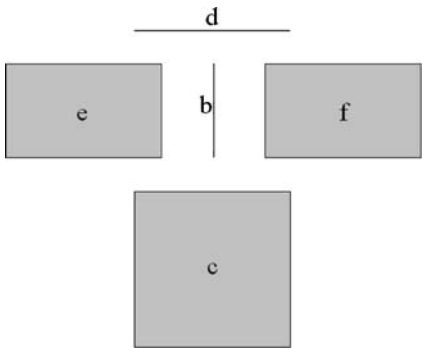

Fig. 1 Sample block world

$\{line(X), medium(X), wide(X)\}, \{line(X), high(X), wide(X)\}\}$
NotConstraints $=$
$\{\{block(X), medium(X), wide(X)\}, \{block(X), high(X), wide(X)\},$
$\{line(X), low(X), wide(X)\}, \{line(X), medium(X), narrow(X)\}\}$
All other possible extensions of binary non-constraints are trivial.

- 4-tuples of unary predicates: all 4-tuples obtained extending ternary non-constraints are trivial, thus in this step both Constraints and NotConstraints are empty. As a side effect, there are no non-constraints of cardinality 4 to be extended, and hence no constraints of cardinality larger than 4 can be found.

– *Step 2*:
- Reflexivity:
NotConstraints $= \emptyset$
Constraints $= \{\{part\_of(X, X)\}, \{on\_top(X, X)\}, \{to\_right(X, X)\}\}$
- Symmetry:
NotConstraints $= \emptyset$
Constraints $=$
$\{\{part\_of(X, Y), part\_of(Y, X)\}, \{on\_top(X, Y), on\_top(Y, X)\},$
$\{to\_right(X, Y), to\_right(Y, X)\}, \{part\_of(X, Y), on\_top(Y, X)\},$
$\{part\_of(X, Y), to\_right(Y, X)\}, \{on\_top(X, Y), to\_right(Y, X)\}\}$
- Transitivity:
NotConstraints $= \{\{on\_top(X, Y), on\_top(Y, Z)\},$
$\{to\_right(X, Y), to\_right(Y, Z)\}, \{part\_of(X, Y), on\_top(Y, Z)\},$
$\{part\_of(X, Y), to\_right(Y, Z)\}, \{to\_right(X, Y), on\_top(Y, Z)\}\}$
Constraints $=$
$\{\{part\_of(X, Y), part\_of(Y, Z)\}, \{on\_top(X, Y), part\_of(Y, Z)\},$
$\{to\_right(X, Y), part\_of(Y, Z)\}, \{on\_top(X, Y), to\_right(Y, X)\}\}$
- Convergence:
NotConstraints $=$
$\{\{on\_top(X, Y), on\_top(Z, Y)\}, \{on\_top(X, Y), part\_of(Z, Y)\},$
$\{to\_right(X, Y), part\_of(Z, Y)\}, \{on\_top(X, Y), to\_right(Z, Y)\}\}$
Constraints $=$
$\{\{part\_of(X, Y), part\_of(Z, Y)\}, \{to\_right(X, Y), to\_right(Z, Y)\}\}$

– Divergence:
NotConstraints $= \{\{part\_of(X,Y), part\_of(X,Z)\}, \{on\_top(X,Y), on\_top(X,Z)\}, \{on\_top(X,Y), to\_right(X,Z)\}\}$
Constraints $= \{\{to\_right(X,Y), to\_right(X,Z)\}, \{on\_top(X,Y), part\_of(X,Z)\}, \{to\_right(X,Y), part\_of(X,Z)\}\}$;

– *Step 3*: There are no non-constraints concerning reflexivity and symmetry to be extended, and 12 non-constraints coming from transitivity, convergence and divergence that can be extended to obtain new integrity constraints. Each of such non-constraints, let us call it C, has cardinality 2, so it can be extended by adding at most 2 more predicates (since $N = 4$), and contains three variables (X, Y, Z), so the only possibilities to be checked are:
  a. adding a unary predicate to one variable:
  $C \cup \{block(X)\}, C \cup \{line(X)\}, C \cup \{low(X)\}, C \cup \{medium(X)\}$,
  $C \cup \{high(X)\}, C \cup \{narrow(X)\}, \{wide(X)\}$ and similarly for Y and Z
  b. adding a unary predicate to two variables:
  $C \cup \{block(X), block(Y)\}, C \cup \{block(X), line(Y)\}, C \cup \{block(X), low(Y)\}$,
  $C \cup \{block(X), medium(Y)\}, C \cup \{block(X), high(Y)\}$,
  $C \cup \{block(X), narrow(Y)\}, C \cup \{block(X), wide(Y)\}$
  $C \cup \{block(X), block(Z)\}, C \cup \{block(X), line(Z)\}, C \cup \{block(X), low(Z)\}$,
  $C \cup \{block(X), medium(Z)\}, C \cup \{block(X), high(Z)\}$,
  $C \cup \{block(X), narrow(Z)\}, C \cup \{block(X), wide(Z)\}$
  $C \cup \{block(Y), block(Z)\}, C \cup \{block(Y), line(Z)\}, C \cup \{block(Y), low(Z)\}$,
  $C \cup \{block(Y), medium(Z)\}, C \cup \{block(Y), high(Z)\}$,
  $C \cup \{block(Y), narrow(Z)\}, C \cup \{block(Y), wide(Z)\}$
  (excluding those that are a superset of constraints found in step **a.**);
  c. adding a binary non-constraint on unary predicates to one variable:
  $\{ C \cup \{block(X), medium(X)\}, C \cup \{block(X), high(X)\}$,
  $C \cup \{block(X), wide(X)\}, C \cup \{line(X), low(X)\}, C \cup \{line(X), medium(X)\}$,
  $C \cup \{line(X), narrow(X)\}, C \cup \{line(X), wide(X)\}, C \cup \{low(X), wide(X)\}$,
  $C \cup \{medium(X), narrow(X)\}, C \cup \{medium(X), wide(X)\}$,
  $C \cup \{high(X), wide(X)\}$ } and similarly for Y and Z.
  (excluding those that are superset of constraints found in step **a.**).

3.2 Descriptors type domains and abducibles

At the end of the procedure reported in Algorithm 3.1, the set of constraints of cardinality 2 can be input to the type induction procedure presented in [13] in order to infer type domains. Then, all pairs of unary predicates belonging to the same domain can be eliminated from the set $Constraints$, thus reducing the complexity of the abductive proof procedure, and a new kind of constraint will be introduced to represent types, such that no two values from the same type domain will be allowed applied to the same object. For example, if the descriptor type domain for the color property is $\{blue, red, yellow, black, green\}$, and the object X is part

of an observation, it will be impossible to abduce two different color descriptors from the above set applied to X.

The whole strategy for inducing the descriptors type domains, sketched in Algorithm 3.3, is now summarized.

Algorithm 3.3 Identification of type domains

Require: Description language L

$U := \{p \in L \mid p \text{ unary}\}$
$E := \{(p, q) \in U \times U \mid \nexists X : p(X) \wedge q(X)\}$
$G_e := (U, E)$
$S := \{C \subseteq U \mid C \text{ clique in } G_e\}$
$F := \{(p, q) \in S \times S \mid p \cap q = \emptyset\}$
$G_d := (S, F)$
$T := \{C \subseteq S \mid C \text{ clique in } G_d\}$
return $argmax_{t \in T}(|\bigcup_{t_i \in t} t_i|)$

The first consideration one can do is that different values for the same attribute are mutually exclusive, since one given object cannot have two of them at the same time. Hence, the first problem to be solved is finding all couples of predicates that are mutually exclusive, i.e. never co-occur referred to the same object in the available knowledge of the world.

It goes without saying that finding mutually exclusive couples is not sufficient: More precisely, *any* value in a given domain cannot co-occur in one object with *any* other value in the same domain. Thus, the problem becomes identifying groups of unary predicates whose elements are *couplewise* mutually exclusive. In particular, since for any set of predicates fulfilling such property it holds that all of its subsets fulfill the same property as well, we are interested in maximal sets only, i.e., we discard groups that are subsets of other groups. This can be obtained by mapping the problem onto a corresponding one in the graph context. Specifically, we build an undirected graph G_e whose nodes are unary predicates in the description language, and where an edge connects two nodes if and only if they are mutually exclusive. Here, the maximal sets we are looking for correspond to all the *maximal* cliques (i.e., cliques that cannot be further extended) in G_e.

The groups found this way are still far from being the desired solutions. Indeed, there can be groups of predicates with couplewise mutually exclusive elements even if they do not refer to a same attribute. For instance, it is generally true that a line is never too tall, hence in a paper document domain we might find the group {*line*, *high*, *very_high*, *highest*} in which it is obvious that value *line* belongs to the domain of type *shape*, while the other three values refer to the type *height*. Nevertheless, we expect that two correct (i.e., distinct, or, more precisely, *disjoint*) groups exist, one containing all (and only those) values belonging to property *shape*, and the other containing all (and only those) values belonging to property *height*. Here, the clue is that, in the end, the desired solution will include only groups that have no element in common. Hence, since the above group would have elements in common with properties *height* and *shape*, it should be discarded. Again, this problem can be solved in the graph context by building an undirected graph G_d in which nodes are groups identified in the previous step as cliques of graph G_e, and an edge connects two nodes if and only if they are disjoint sets. Now, the solution will be made up by only couplewise disjoint subsets,

and specifically by maximal groups of disjoint subsets, each of which corresponds to a maximal clique in G_d.

However, the clique in G_d will probably not be unique, in which case one must have a clue for choosing the right one. The intuition, in this case, is that any "wrong" clique, in order to fulfill the mutual disjunction requirement, will have overall a number of values that is less than that of the correct solution, since the correct solution should be the only one containing all the possible values for each property (represented by a group), and hence the union of predicates in all of its components should be equal to the whole set of values for all possible attributes. In other words, the solution is actually a *partition* of the set of unary predicates. This holds because the description language is assumed not to contain "boolean" properties; if it does, the union would not be a partition, but should in any case contain more unary predicates than any other candidate partition.

It is worth noting that the feasibility of reaching the target solution requires that the number of values for the domains to be identified and the amount of available knowledge about observations to be strictly proportional. Indeed, the more the values, the more the possible interrelations that can take place between them. If the available observations are not sufficiently significant, i.e., too many existing interrelations are not recognizable in them, then knowledge about the actual biases in the given domain would be too loose for the algorithm to properly separate semantically different values.

4 Experiments

The proposed methods were tested on various domains (Scientific Paper Domains [11], Family Relationships [1], Multiplexer [27] and Congressional Votes [18]) suitably chosen in order to cover all the possible cases of available observations and target types to be recognized. In the following we show both the experiments aimed at learning the descriptors type domain when imperfect data are provided and other experiments showing the benefit that the learning process can bring by the exploitation of the learned abductive theory.

4.1 Descriptors type domains and abducibles

The first experiment concerns the Scientific Papers dataset. It is based on a representation language made up of predicates with various arities, of which unary predicates represent values belonging to many different domains (*general* case). It is made up of 112 scientific papers, belonging to 4 different classes, whose layout structure was described in terms of its composing layout blocks features (height, width, horizontal position, vertical position, content type) and relative position (horizontal adjacency, vertical adjacency, horizontal alignment, vertical alignment). The procedure (see Algorithm 3.3) found the following (correct) types:

1. *Width*: {large, medium, medium_large, medium_small, small, very_large, very_small}
2. *Content*: {graphic, hor_line, image, mixed, text,ver_line}

3. *Vertical position*: {lower, middle, upper}
4. *Horizontal position*: {center, left, right}
5. *Height*: {large, medium, medium_large, medium_small, small, smallest, very_large, very_small, very_very_large, very_very_small}

The Family Relationships dataset [1] refers to a description language made up of predicates with various arities, of which unary predicates all belong to the same type. It describes a hypothetical family in terms of each person's sex and of the basic relations among persons (parent and married), whose members' pairs are tagged according to the derived relations (father, mother, son, daughter, uncle, aunt, etc.). In this case, all the unary predicates fell in one group (thus there was no need for building G_d), that was also the only type (successfully retrieved by the algorithm):

1. *Sex*: {female, male}

The Multiplexer dataset [27] describes 6-bit configurations, with the aim of inducing the definition of a multiplexer such that, among the last four bit positions, the position denoted by the first two bits must be 1. All 64 possible bit configurations are included, which should make significantly easier the type induction task, as confirmed by the algorithm output:

1. *Sixth bit*: {bit6at0, bit6at1}
2. *Fifth bit*: {bit5at0, bit5at1}
3. *Fourth bit*: {bit4at0, bit4at1}
4. *Third bit*: {bit3at0, bit3at1}
5. *Second bit*: {bit2at0, bit2at1}
6. *First bit*: {bit1at0, bit1at1}

Lastly, the Congressional Votes [18] dataset describes 435 Congressmen as being democrats (267) or republicans (168) according to their votes on 16 issues. The 435 examples are described by means of 32 predicates each representing the favorable (*y*) or opposite (*n*) vote on one of the above issues. It is particularly interesting because a certain amount of noise is present in the descriptions, in the form of unknown (omitted) votes, as reported in Table 1. Nevertheless, the algorithm is able to correctly infer all the 16 types (corresponding to the issues), each with its 2 descriptors (corresponding to the yes/no options).

Now, our aim is to check the effectiveness of the proposed procedure in handling imperfect data. To this purpose, we focused on the Scientific Papers dataset, for a number of reasons. First, because it is a real-world one, and is probably the most complex among those considered. Second, the shape of the descriptions is not fixed, differently from the Votes and Multiplexer ones. Third, it was made up of many different observations, differently from the Family one. Various experiments were run, in which noise was progressively introduced in the dataset descriptions. For each fixed amount of noise to be introduced, 10 random corruptions of the dataset were performed, on which running the proposed algorithm. Then, the learned types were checked and categorized in one of the following categories (listed by decreasing desirability): *correct*, *incomplete* (i.e., missing some types or some values in some type domains, but without mixing values belonging to different types), *impossible* (when the algorithm autonomously recognized that the available information was too loose for getting to a correct solution), and *wrong* (when at least one of the identified types

Table 1 Noise on congressmen votes

Issue	Omissions
Handicapped infants	0
Crime	25
Adoption budget resolution	48
mx missile	15
Physicians fee freeze	11
el salvador aid	11
Religious groups in schools	15
Immigration	22
Synfuels corporation cutback	7
Education spending	21
Water project cost sharing	12
Duty free exports	17
Aid to nicaraguan contrast	14
Superfund right to sue	31
Export administration act S.A.	28
Anti satellites test ban	11

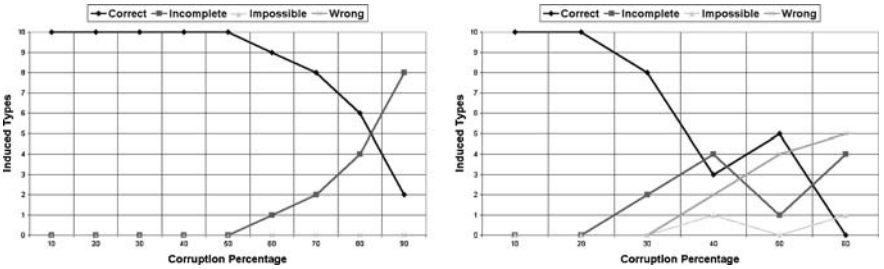

Fig. 2 Descriptors type domains: progressively smaller datasets (*left*) and progressively incomplete descriptions (*right*)

contained in its domain values actually belonging to different types). A first experiment in this direction aimed at assessing how sensitive the algorithm is to the amount of observations provided to it. In this case, the dataset corruption consisted in progressively eliminating observations (examples) from it (remember that the initial size was 112). The amount of corruption ranged between 10 and 90% of the entire dataset, and the corresponding results are reported in Fig. 2. It is interesting to note that the algorithm never generated undesirable (i.e., impossible or wrong) type domains. Actually, up to 50% of the dataset it always gave correct and complete answers. After that threshold, completeness started decreasing, but even when 90% of the observations was dropped (i.e., only 12 paper descriptions were available) in two cases it succeeded in finding the correct and complete types. This should allow one to state that the system is effective also when provided with very few observations.

Then, the next question was how much noise could be present in the available knowledge in order for the system not to be misled in its task. For this purpose, each available observation was corrupted by progressively eliminating a portion from 10% up to 60% of its description. The experimental outcomes,

graphically represented in Fig. 2, suggest that the algorithm is more sensitive to partial descriptions than it was to a small number of observations. Indeed, in this case complete and correct types are induced only up to 20% of corruption. Considering as a good outcome also incomplete types raises the threshold up to 30%. Anyway, also for more dramatic corruptions, the desirable (i.e., correct + incomplete) outcomes far outperform the undesirable ones. Only when 60% of each description in the dataset is dropped the number of wrong inductions becomes predominant, but interestingly it does not exceed half of the trials.

This behavior can be explained because the proposed algorithm for the descriptor type domains heavily relies on co-occurrence of values for inducing the type domains. Thus, eliminating whole observations, but leaving complete the remaining ones, potentially still preserves many co-occurrences. On the contrary, dropping portions of each observation is likely to introduce false (supposed) incompatibilities among values that actually belong to different types. As already pointed out, some of these false incompatibilities are already present in the complete dataset (e.g., a line can have any width or height but is never too thick), thus artificially adding more noise of this kind makes an already hard task even harder. However, if the procedure is to be used in a Machine Learning context, incomplete (unknown) information in the available observations is a problem on its own, and experimental results, reported in the following, show that abductive operators can cope with it only to some extent, which is in any case far below the threshold after which the proposed algorithm's performance becomes too low to be acceptable (and in general does not deal with datasets in which all descriptions are corrupted).

4.2 Exploitation of the learned abductive theories

This section reports the experiments carried out on the same datasets reported in the previous section exploiting the *abducibles* and the *integrity constraints* automatically learned by the procedures presented in Sect. 3. INTHELEX [9], an incremental inductive logic programming system, has been provided with the abductive proof procedure [12] in order to complete the observations in such a way that the corresponding examples are correctly classified by the already generated theory, thus avoiding, whenever possible, the use of the operators to modify the theory.

4.2.1 Multiplexer

The "multiplexer" problem aims at learning the definition of a 6-bits multiplexer. The dataset contains descriptions of all possible configurations of 6-bits, in which the first 2-bits represent the address of one of the subsequent 4-bits, that must be set at 1. Thus, if the bit addressed is actually 1 the example is positive, otherwise it is considered as negative for the target concept.

Example 4.1 The multiplexer configuration `010110` contains, in the first and second position, the pair 01, that is the binary representation of the decimal number 2; thus, it addresses the second element of the 4-tuple 0110 (i.e., the remaining

```
mul(X) :-      mul(X) :-      mul(X) :-      mul(X) :-
  bit1at0(X),    bit1at0(X),    bit1at1(X),    bit1at1(X),
  bit2at0(X),    bit2at1(X),    bit2at0(X),    bit2at1(X),
  bit3at1(X).    bit4at1(X).    bit5at1(X).    bit6at1(X).
```

Fig. 3 A correct definition for multiplexer configurations

part of the original configuration). Since the addressed bit is 1, the configuration description represents a positive example.

Since a 6-bits multiplexer can assume $2^6 = 64$ possible configurations, the complete training set is made up of 64 examples, 32 positive and 32 negative. The representation language of the observations is the same as in [27]: For instance, the multiplexer `100110` is represented by the clause

```
mul(e) :- bit1at0(e), bit2at1(e), bit3at0(e),
bit4at1(e), bit5at1(e), bit6at0(e)
```

The first step was checking if the system is actually able to learn the expected theory. To this purpose, it was provided with a complete training set containing all the 64 possible configurations. Starting from scratch, the resulting learned theory (see Fig. 3) was composed of four clauses describing the multiplexer problem. It was obtained, in 1.38 s, performing 12 theory revisions.

Successively, an incomplete dataset was obtained by corrupting 12 examples out of 64 so that only three bits out of six of the original configuration were specified. Both the examples to be corrupted and their bits to be neglected were randomly selected for 10 times.

Example 4.2 Suppose that the configuration in the previous example, 010110, is corrupted by omitting the second, third and sixth bits. Now, the resulting configuration is 0?0?1? and its representation is

```
mul(e) :- bit1at0(e), bit3at0(e), bit5at1(e).
```

As described in [27], such an incomplete dataset was exploited for learning theories in two different ways: first using induction only, and then using induction supported by abduction. The theories obtained in the two cases were tested (without using abduction) on the uncorrupted dataset. Table 2 shows the system performance in the two cases, averaged on the 10 corrupted datasets, as regards the number of definitions in the learned theories, the performed theory revisions, the number of exceptions, runtime and predictive accuracy (i.e., number of examples correctly classified/total number of examples). The learned Abduction Theory provided to the system included all the predicates of the form `bitNatB` ($N \in \{1..6\}$, $B \in \{0, 1\}$) as abducibles, and integrity constraints of the

Table 2 System performance on the multiplexer dataset

Data	Definitions	Revisions	Exceptions	Runtime (s)	Accuracy
Without abduction	4.1	6.05	2.05	4.55	99.38
With abduction	4.1	5.55	0.4	4.36	99.22

Table 3 System performance on the congressional voting records dataset

Data	Definitions	Revisions	Exceptions	Runtime (s)	Accuracy
Without abduction	12.40	26.90	1.7	30.30	93.33
With abduction	10.10	19.20	0.80	41.36	96.8

form `ic[(bitNat0(X), bitNat1(X))]` (meaning that "if the bit in position *N* is set to 0 it can't be set to 1, and *vice versa*").

Our expectation was that, when exploiting abduction, the system should be able to do correct assumptions on the omitted bits value and position, thus recovering from the missing information. Indeed, it was able to capture the correct definitions (the same as those shown in Fig. 3) but applying less theory revisions, adding less exceptions and in a shorter execution time. In particular, it is noteworthy the decrease in the number of exceptions with respect to those that induction alone needed in order to account for the learned theory.

Example 4.3 In one of the 10 sets containing the corrupted example

```
mul(e16) :- bit1at0(e16),bit5at1(e16),bit6at1(e16).
```

The abduction succeeded in completing the available description, thus avoiding a revision of the theory, by making the following assumptions: bit3at1(e16), bit2at0(e16). Note that no assumption was made on the status of bit 4, that is not significant with respect to the address 0 (corresponding to bit 3) expressed by the first two bits.

4.2.2 Congressional voting records

On the Multiplexer dataset, the use of abduction in support of induction succeeded in improving the system performance as regards both the number of revisions/exceptions and the runtime, but not concerning the predictive accuracy. Here, an experiment showing that abduction could improve accuracy, as well, is presented. The problem, as reported in [18], consists in classifying a Congressman as a democrat (target concept) or a republican (*not(democrat)*), according to his votes on the 16 issues in Table 1. A certain amount of noise is present in the descriptions, in the form of `unknown` votes, that were omitted, resulting in a distribution for each issue (Table 1).

Definitions for the class *democrat* were learned, exploiting first pure induction and then induction plus abduction, starting from the empty theory. The corresponding predictive accuracy was tested, according to a 10-fold cross validation methodology, ensuring that each fold contained the same proportion of positive and negative examples. Table 3 shows the system performance on this dataset. It is possible to note that the use of abduction improves all evaluation parameters, except Runtime. This can be explained by taking into account the additional time needed to search for consistent abductive explanations due to the large number of integrity constraints in the learned abductive theory.

4.2.3 Family relationships

So far, the presented experiments were carried out on datasets whose incompleteness was fixed. Now, a new experiment is described, whose aim was investigating

the abductive proof procedure behavior with respect to different degrees of incompleteness. In this case, we followed the same approach adopted by [20] on the same dataset, not only as regards the corruption of the available family description, but also concerning other problem settings. First of all, only examples about *father* were taken into account: the training set included 36 positive examples and 200 negative ones that were randomly generated. Moreover, the examples description is more complex than before, in that it includes not only the basic observations (*male*, *female*, *parent*, *married*) but also all the known facts concerning the concepts other than *father* (i.e., *son*, *daugther*, *mother*, etc.), for a total of 742 literals. Progressive corruption of such a complete description was obtained by randomly eliminating facts from it. Specifically, learning was run on the following percentages of preserved descriptions: 100% (no incompleteness), 90, 80, 70, 60, 50, and 40%. Hence, the description size varied as follows: 742 literals (100%), 668 literals (90%), and so on. For each percentage, the dataset was corrupted in five different ways, thus obtaining five corresponding learning problems whose performance was averaged according to a five-fold cross validation methodology, ensuring that each fold contained the same proportion of positive and negative examples.

Here an extract of the learned Abduction Theory, specifically some *integrity constraints*, for this domain is reported (whose interpretation is: "one person cannot be both male and female"; "a son cannot be female, and *vice versa*"; "a daughter cannot be male, and *vice versa*"):

```
ic([male(X),female(X)]). ic([son(X,Y),female(X)]).
ic([daughter(X,Y),male(X)]).
```

Comparing the performance with and without abduction on the corrupted datasets, the benefit becomes very evident with respect to all the parameters taken into account in Table 4: number of definitions, number of theory revisions, runtime and predictive accuracy. It is possible to note how abduction is able to preserve the theories from being refined (indeed, the number of revisions per clause dramatically decreases). Moreover, lower runtimes (except in one case) prove that the abductive procedure is also efficient. Finally, note that, in spite of the number of clauses being less when using abduction in all corrupted cases, predictive accuracy is always higher than the case without abduction.

4.2.4 Scientific paper domain

This section presents the experiments concerning the induction of classification rules for a dataset obtained by corrupting a subset of the scientific paper documents belonging to one of the four classes. The corruption consisted in eliminating 8% of the descriptors for each observation (made up of 112 facts on average (76 min.–170 max.)) contained in the tuning set. INTHELEX was applied first without exploiting its abductive procedure. Successively, the learning process was repeated, allowing the system to exploit its abductive capability, after learning the abducibles and the integrity constraints according to the procedures described in Sect. 3. We focused our attention on binary constraints made up of unary and binary predicates, i.e., of the form $(ic([a(X), b(X)], ic([c(X, Y), d(X, Y)])$. Furthermore, another experiment was run to test the usefulness of the descriptors type

Table 4 System performance on the family dataset

(%)		Definitions	Revisions	Revisions/definitions	Runtime	Accuracy
100	noabd	1	1.6	1.6	52.25	99.58
	abd	1	1.2	1.2	47.13	100
90	noabd	2.8	6.2	2.2	146.19	96.28
	abd	1	1.2	1.2	69.04	99.17
80	noabd	3.8	8.8	2.3	190.12	96.27
	abd	1	1.2	1.2	70.35	100
70	noabd	5	9	1.8	218.03	93.78
	abd	1	1.2	1.2	59.70	100
60	noabd	6.8	11.4	1.7	287.57	92.13
	abd	1.8	1	0.5	448.82	100
50	noabd	7.2	9.6	1.3	256.91	92.15
	abd	1.8	1	0.5	43.08	100
40	noabd	10.2	12	1.2	871.51	90.9
	abd	1.8	1	0.5	24.32	98.75

Table 5 System performance on the scientific papers domain

		With abduction	
	Without abduction	Without type domains	With type domains
Revisions	7.72	5.48	1
Clauses	4.09	3.18	1.72
Accuracy (%)	96.24	99.32	98.75
Runtime (s)	5.16	40.05	24.29

domains in discarding some integrity constraints made up of predicates belonging to different type domains.

Table 5 reports the system performance when using the abductive procedure (with and without the exploitation of the learned descriptors type domains) and when not using it, as to the performed theory revisions, the added definitions, the predictive accuracy and the execution time (s). Predictive accuracy and number of theory revisions improve when the abductive procedure is exploited, both with and without the use of descriptors type domains. In particular, the number of theory revisions decreases when type domains are exploited. This means that the system was able to correctly complete the corrupted observations without applying the refinement procedure. As regards runtime, it increases because of the abductive procedure; in this case we can note that when we exploit the description type domains in the choice of the integrity constraints the runtime is better than the case when they were not exploited because less integrity constraints must be taken into account.

4.2.5 Comparison

The proposed approach does not aim at completing the training data before the learning process starts. Rather, its aim is the automatic learning of specific knowledge: abducibles, i.e., the predicates on which hypothesis can be done, and integrity constraints, that are exploited to dynamically handle the incomplete information within the learning process. Thus, a comparison with systems that propose to overcome the problem of handling missing values by pre-processing

Table 6 Comparison of abduction on the family dataset

	100%	90%	80%	70%	60%	50%	40%
INTHELEX	1	99.17	1	1	1	1	98.75
ACL1	1	1	99.60	1	1	97.20	97.60
mFOIL	1	99.20	98.40	97.50	98.40	98.40	95.10

the training data before the learning process starts (FOIL [22], LINUS, ASSISTANT) would be unfair. Nevertheless, we compare the results obtained by the ILP incremental learning system INTHELEX, in which the framework integrating abductive and inductive reasoning is developed, with ACL1 [20] and mFOIL [22], the FOIL extension able to deal with incomplete data. Specifically, we performed the comparison on the family and congressional votes datasets that are the same exploited by [20] for the same purpose. Table 6 reveals that the predictive accuracy results of the comparison on the family dataset for progressive corruption is almost the same as that obtained by the other systems. As regards the experiment on congressional voting, INTHELEX turned out to be absolutely better with respect to the other systems, reporting 96.8% accuracy against 85.25% for ACL1 and 78% for mFOIL. This assesses the effectiveness of the abductive theory automatically learned and exploited in it.

5 Conclusion

Real-world domains are often affected by noise, that makes significantly more difficult the process of knowledge discovery in such environments. As a solution, machine learning studies have moved towards more expressive representations, which in turn require more complex inference mechanisms, such as abductive reasoning. However, such mechanisms strongly rely on the availability of a deep background knowledge about the specific application domain, that is usually provided by a domain expert. Due to the difficulty in formalizing such knowledge, and considering its importance for making the learning process effective, a step forward consists in automatically learning such a theory starting from the available observations on the application domain in order to make the learning system completely autonomous. This paper proposed a methodology for automatically inferring the meta-knowledge to perform the abductive reasoning starting from the available observations. Experiments confirm that the abductive theories automatically inferred are actually capable of improving various parameters of the learning process in different artificial and real-world domains.

References

1. Blockeel H, De Raedt L (1996) Inductive database design. In: Proceedings of the 10th international symposium on methodologies for intelligent systems (ISMIS96), vol 1079 of Lecture Notes in Artificial Intelligence, Springer-Verlag, pp 376–385
2. Cestnik B, Kononenko I, Bratko I (1987) Assistant 86: A knowledge-elicitation tool for sophisticated users. In: Proceedings of EWSL, Sigma Press. Bled, Yugoslavia, pp 31–45
3. Clark K (1978) Negation as failure. In: Gallaire H, Minker J (eds) Logic and databases, Plenum Press, New York, pp 293–322

4. Clark P, Boswell R (1991) Rule induction with CN2: Some recent improvements. In: Proceedings of the fifth European working session on learning, Springer, Berlin Heidelberg New York, pp 151–163
5. Cohen P, Feigenbaum E (eds) (1981) The Handbook of artificial intelligence. vol 3. Morgan Kaufmann, San Mateo, CA
6. De Raedt L (1992) Interactive theory revision—an inductive logic programming approach. Academic Press, New York
7. Dimopoulos Y, Kakas A (1996) Abduction and learning. In: Raedt LD (ed) Advances in inductive logic programming, IOS Press, pp 144–171
8. Eshghi K, Kowalski R (1989) Abduction compared to negation by failure. In: Levi G, Martelli M (eds) Proceedings of the 6th international conference on logic programming, The MIT Press, Cambridge, MA, pp 234–255
9. Esposito F, Ferilli S, Fanizzi N, Basile T, Di Mauro N (2003) Incremental multistrategy learning for document processing. Appl Artif Intell: An Int J 17(8–9):859–883
10. Esposito F, Lamma E, Malerba D, Mello P, Milano M, Riguzzi F, Semeraro G (1996) Learning abductive logic programs. In: Proceedings of the ECAI96 workshop on abductive and inductive reasoning, Budapest, Hungary, pp 23–30
11. Esposito F, Malerba D, Lisi F (2000a) Machine learning for intelligent processing of printed documents. J Intell Inf Syst 14(2–3):175–198
12. Esposito F, Semeraro G, Fanizzi N, Ferilli S (2000b) Multistrategy theory revision: induction and abduction in INTHELEX. Machine Learn 38(1–2):133–156
13. Ferilli S, Esposito F, Basile T, Di Mauro N (2004) Automatic induction of first-order logic descriptors type domains from observations. In: Camacho R, King RD, Srinivasan A (eds) ILP, vol 3194 of LNCS, Springer, Berlin Heidelberg New York, pp 116–131
14. Flach P, Lachiche N (2001) Confirmation-guided discovery of first-order rules with `Tertius`. Machine Learn 42(1–2):61–95
15. Hewett R, Leuchner J (2002) Knowledge discovery with second-order relations. Knowledge Inf Syst 4(4):413–439
16. Hinneburg A, Keim D (2003) A general approach to clustering in large databases with noise. Knowledge Inf Syst 5(4):387–415
17. Kakas A, Mancarella P (1990) On the relation of truth maintenance and abduction. In: Proceedings of the 1st pacific rim international conference on artificial intelligence, Nagoya, Japan
18. Kakas A, Riguzzi F (1999) Abductive concept learning. New Gen Comput
19. Kakas A, Kowalski R, Toni F (1993) Abductive logic programming. J Logic Comput 718–770
20. Kakas C, Riguzzi F (2000) Learning with abduction. New Gen Comput 18(3):243–294
21. Lamma E, Mello P, Milano M, Riguzzi F, Esposito F, Ferilli S, Semeraro G (2000) Cooperation of abduction and induction in logic programming. In: Kakas A, Flach P (eds) Abductive and inductive reasoning: essays on their relation and integration, Kluwer, Dordrecht
22. Lavrač N, Džeroski S (1994) Inductive logic programming: techniques and applications. Ellis Horwood, New York
23. Michalski R (1994) Inferential theory of learning. developing foundations for multistrategy learning. In: Michalski R, Tecuci G (eds) Machine learning. A multistrategy approach, vol IV. Morgan Kaufmann, San Mateo, CA, pp 3–61
24. Poole D (1988) A logical framework for default reasoning. Artif Intell 36:27–47
25. Raedt LD, Dehaspe L (1997) Clausal discovery. Machine Learn 26(2):99–146
26. Reiter R (1980) A logic for default reasoning. J Artif Intell 13:81–132
27. Riguzzi F (1998) Extensions of Logic Programming as Representation Languages for Machine Learning, PhD thesis, University of Bologna
28. Skillicorn DB, Wang Y (2001) Parallel and sequential algorithms for data mining using inductive logic. Knowledge Inform Syst 3(4):405–421

Author biographies

Floriana Esposito received the Laurea degree in electronic Physics from the University of Bari, Italy, in 1970. Since 1994 is Full Professor of Computer Science at the University of Bari and Dean of the Faculty of Computer Science from 1997 to 2002. She founded and chairs the Laboratory for Knowledge Acquisition and Machine Learning of the Department of Computer Science. Her research activity started in the field of numerical models and statistical pattern recognition. Then her interests moved to the field of Artificial Intelligence and Machine Learning. The current research concerns the logical and algebraic foundations of numerical and symbolic methods in machine learning with the aim of the integration, the computational models of incremental and multistrategy learning, the revision of logical theories, the knowledge discovery in data bases. Application include document classification and understanding, content based document retrieval, map interpretation and Semantic Web. She is author of more than 270 scientific papers and is in the scientific committees of many international scientific Conferences in the field of Artificial Intelligence and Machine Learning. She co-chaired ICML96, MSL98, ECML-PKDD 2003, IEA-AIE 2005, ISMIS 2006.

Stefano Ferilli was born in 1972. After receiving his Laurea degree in Information Science in 1996, he got a Ph.D. in Computer Science at the University of Bari in 2001. Since 2002 he is an Assistant Professor at the Department of Computer Science of the University of Bari. His research interests are centered on Logic and Algebraic Foundations of Machine Learning, Inductive Logic Programming, Theory Revision, Multi-Strategy Learning, Knowledge Representation, Electronic Document Processing and Digital Libraries. He participated in various National and European (ESPRIT and IST) projects concerning these topics, and is a (co-)author of more than 80 papers published on National and International journals, books and conferences/workshops proceedings.

Teresa M.A. Basile got the Laurea degree in Computer Science at the University of Bari, Italy (2001). In March 2005 she discussed a Ph.D. thesis in Computer Science at the University of Bari titled "A Multistrategy Framework for First-Order Rules Learning." Since April 2005, she is a research at the Computer Science Department of the University of Bari working on methods and techniques of machine learning for the Semantic Web. Her research interests concern the investigation of symbolic machine learning techniques, in particular of the cooperation of different inferences strategies in an incremental learning framework, and their application to document classification and understanding based on their semantic. She is author of about 40 papers published on National and International journals and conferences/workshops proceedings and was/is involved in various National and European projects.

Nicola Di Mauro got the Laurea degree in Computer Science at the University of Bari, Italy. From 2001 he went on making research on machine learning in the Knowledge Acquisition and Machine Learning Laboratory (LACAM) at the Department of Computer Science, University of Bari. In March 2005 he discussed a Ph.D. thesis in Computer Science at the University of Bari titled "First Order Incremental Theory Refinement" which faces the problem of Incremental Learning in ILP. Since January 2005, he is an assistant professor at the Department of Computer Science, University of Bari. His research activities concern Inductive Logic Programming (ILP), Theory Revision and Incremental Learning, Multistrategy Learning, with application to Automatic Document Processing. On such topics HE is author of about 40 scientific papers accepted for presentation and publication on international and national journals and conference proceedings. He took part to the European projects 6th FP IP-507173 VIKEF (Virtual Information and Knowledge Environment Framework) and IST-1999-20882 COLLATE (Collaboratory for Annotation, Indexing and Retrieval of Digitized Historical Archive Materials), and to various national projects co-funded by the Italian Ministry for the University and Scientific Research.