Noname manuscript No.
(will be inserted by the editor)

Probabilistic Inductive Constraint Logic

Fabrizio Riguzzi · Elena Bellodi · Riccardo Zese · Marco Alberti · Evelina Lamma

the date of receipt and acceptance should be inserted later

Abstract Probabilistic logical models deal effectively with uncertain relations and entities typical of many real world domains. In the field of Probabilistic Logic Programming (PLP) usually the aim is to learn these kinds of models to predict specific atoms or predicates of the domain, called target atoms/predicates. However, it might also be useful to learn classifiers for interpretations as a whole: to this end, we consider the models produced by the Inductive Constraint Logic (ICL) system, represented by sets of *integrity constraints*, and we propose a probabilistic version of them. Each integrity constraint is annotated with a probability, and the resulting probabilistic logical constraint model assigns a probability of being positive to interpretations. To learn both the structure and the parameters of such probabilistic models we propose the system PASCAL for "ProbAbiliStic inductive ConstrAint Logic". Parameter learning can be performed using gradient descent or L-BFGS. PASCAL has been tested on 11 datasets and compared with a few statistical relational systems and a system that builds relational decision trees (TILDE): we demonstrate that this system achieves better or comparable results in terms of area under the Precision-Recall and Receiver Operating Characteristic curves, in a comparable execution time.

Fabrizio Riguzzi, Marco Alberti
Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
E-mail: [fabrizio.riguzzi,marco.alberti]@unife.it

Elena Bellodi, Riccardo Zese, Evelina Lamma
Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
E-mail: [elena.bellodi,riccardo.zese,evelina.lamma]@unife.it

1 Introduction

Uncertain information is being taken into account in an increasing number of application fields. Probabilistic logical models are a suitable framework to handle uncertain information, but usually require expensive inference and learning procedures. For this reason, in the last decade many languages that impose limitations to the form of sentences have been proposed.

A possible way to pursue this goal is the application of learning from interpretations [15, 7] instead of the classical setting of learning from entailment. In fact, given fixed bounds on the maximal length of clauses and the maximal arity of literals, first-order clausal theories are polynomial-sample polynomial-time PAC-learnable [15]. Moreover, examples in learning from interpretations can be considered in isolation [7], so coverage tests are local and learning algorithms take a time that is linear in the number of examples.

A particularly interesting system that learns from interpretations is Inductive Constraint Logic (ICL) [17], based on the language of Constraint Logic Theories. It performs discriminative learning and it generates models in the form of sets of integrity constraints. It can be seen as the dual of ILP systems based on learning from entailment.

In this paper we propose a probabilistic version of integrity constraints, called Probabilistic Constraint Logic Theories (PCLTs) [1], where each integrity constraint is annotated with a probability. Under our formalism, models assign a probability of being positive to interpretations. This probability can be computed in a time that is logarithmic in the number of groundings of the constraints that are violated in an interpretation.

We also present the system PASCAL for "ProbAbiliStic inductive ConstrAint Logic" that learns both the structure and the parameters of these models. On an experimental level, PASCAL has been compared with: (1) SLIPCOVER, LIFTCOVER and LEMUR, three state-of-art PLP systems, (2) Markov Logic Networks (MLNs) and (3) TILDE, a relational decision tree induction algorithm.

SLIPCOVER [6] performs structure learning of Logic Programs with Annotated Disjunctions (LPADs) [45] using knowledge compilation for parameter learning, by means of the language of Binary Decision Diagrams. The head of each clause in a LPAD is composed of a disjunction of logical atoms, each annotated with the probability of being true when the body holds; probabilities in the head must sum up to 1. LEMUR [19] learns LPADs by means of a Monte Carlo tree search algorithm. LIFTCOVER [32] performs structure learning of liftable PLP programs, characterized by clauses all having a single atom of the same predicate in the head, annotated with a probability. All algorithms are specialized for discriminative learning, i.e. they guarantee good predictions for the so-called *target predicates* (the ones appearing in the clauses' head). TILDE [8] has been adapted to probabilistically classify interpretations.

PASCAL differs from all the previous algorithms in that:

– it learns probabilistic integrity constraints, i.e. clauses which are annotated with a probability *as a whole*, while SLIPCOVER, LIFTCOVER and LEMUR learn probabilistic clauses with annotated heads;
– it performs probabilistic classification of interpretations instead of classifying target atoms (as done by SLIPCOVER, LIFTCOVER, LEMUR and the probabilistic adaptation of TILDE);
– it is based on a more expressive language than the one allowed by SLIPCOVER, LIFTCOVER, LEMUR and TILDE: probabilistic integrity constraints admit every predicate of the domain both for the head and the body, while LPADs clauses admit only target predicates in the head and TILDE clauses only the positive class for the head;
– it encodes a distribution on the class variable given the atom variables.

Due to the first three characteristics above, PASCAL can be seen as the dual of SLIPCOVER/LIFTCOVER/LEMUR.

Results show that PASCAL is able to achieve better or comparable results both in terms of quality of the learnt models (measured by the area under the Precision-Recall and the Receiver Operating Characteristic curves) and learning time with respect to all systems. SLIPCOVER, LIFTCOVER and LEMUR, in turn, were shown to be comparable with state-of-art ILP systems [6,32].

Finally, we believe that PCLTs may be a suitable formalism to introduce probabilistic reasoning in the framework of interaction protocols in societies of agents [2], where the language of Constraint Logic Theories was defined to verify the compliance of interacting agents (or query answering in [22]) to a set of integrity constraints, integrated with a knowledge based expressed as an abductive logic program. PCLTs might allow to monitor and verify, for instance at run-time, the compliance of a partial, and still not complete, interpretation (i.e., a narrative of occurred events, but not yet completed).

The paper is organized as follows: Section 2 introduces integrity constraints and ICL, Section 3 presents probabilistic integrity constraints, Section 4 introduces the parameter learning problem, Section 5 illustrates PASCAL, Section 6 discusses related work, Section 7 describes the experiments performed and Section 8 concludes the paper.

2 Inductive Constraint Logic (ICL)

ICL [17] performs discriminative learning from interpretations. It learns logical theories in the form of Constraint Logic Theories (CLTs).

2.1 Logic Preliminaries

We consider a logic without function symbols so a *signature* is a pair (Σ_c, Σ_p) where Σ_c is a set of *constants*, and Σ_p is a set of *predicate symbols* with arity, containing the equality binary predicate $\approx$.

A first-order *theory* is built upon a signature and a countable set of *variables*. A *term* is a constant, or a variable. An *atom* is a predicate symbol applied to as many terms as the symbol's arity. A *literal* L is either an atom A (also called a *positive literal*) or its *negation* $\neg A$ (a *negative literal*).

A *normal logic program* **BG** is a set of formulas, called *clauses*, of the form

$$H \leftarrow B_1, \ldots, B_n \tag{1}$$

where H is an atom and all the B_is are literals. H is called the *head* of the clause and $B_1, \ldots, B_n$ is called the *body*. If the body is empty the clause is called a *fact*.

A term, atom, literal or clause is *ground* if it does not contain variables. A *substitution* θ is an assignment of variables to terms: $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$. The *application of a substitution* $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$ *to a term, atom, literal or clause* r, indicated with $r\theta$, is the replacement of each variable V_i appearing in r and in θ with t_i. $r\theta$ is called an *instance* of r. θ is a *grounding* for r if $r\theta$ is ground.

The semantics of first-order formulas is given by providing interpretations for the constant and predicate symbols over a universe of individuals. We consider *Herbrand interpretations*, whose universe are the ground terms of the language. The Herbrand base of a language is the set $\mathcal{B}_H = \{p(t_1, \ldots, t_n) | p$ is a predicate symbol of arity n and $t_1, \ldots, t_n$ are ground terms$\}$.

A Herbrand interpretation for a theory is a subset of the Herbrand base for the language built over the constant and predicate symbols that occur in the theory; the atoms included in an interpretation are *true in the interpretation*. Given an interpretation I, an atom A is true in I, written $I \models A$, if $A \in I$, and the negation of an atom $\neg A$ is true in I, written $I \models \neg A$, if $A \notin I$.

Given a logic program, there are various ways to assign it a meaning, corresponding to different semantics. A semantics associates a program with a model or a set of models, usually Herbrand interpretations. We consider here the Clark's completion semantics [12] that assigns a Herbrand interpretation to a program. We indicate such an interpretation for a normal program **BG** with $M(\mathbf{BG})$.

2.2 ICL

A Constraint Logic Theory (CLT) is a set of *integrity constraints*. In the following, we recall the definition of an integrity constraint from [28].

An integrity constraint (IC) is a formula C of the form

$$L_1, \ldots, L_b \rightarrow A_1; \ldots; A_h \tag{2}$$

where each L_i is a logical literal (i.e., a logical atom or the negation of a logical atom) and each A_j is a logical atom. $L_1, \ldots, L_b$ is called the *body* of C ($Body(C)$) and $A_1; \ldots; A_h$ is called the *head* of C ($Head(C)$).

The semantics of ICs is based on interpretations as in First-Order Logic (FOL). We now define the truth of an integrity constraint in an interpretation.

An IC C is *true in an interpretation* I ($I \models C$) if and only if, for each grounding substitution θ such that each literal in $Body(C)\theta$ is true in I, at least one atom in $Head(C)\theta$ is true in I. Thus, the body of an IC is read as a conjunction and its head as a disjunction.

A CLT can be complemented with a normal logic program **BG** expressing background knowledge about a domain.

With a slight abuse of notation, we indicate with $\mathbf{BG} \cup I$ the normal logic program composed of **BG** and the fact $A_i \leftarrow$ for each atom $A_i \in I$. We indicate $\mathbf{BG} \cup I$'s model according to Clark's completion semantics ([12]) by $M(\mathbf{BG} \cup I)$. Intuitively, $M(\mathbf{BG} \cup I)$ is I augmented by the atoms that can be derived by **BG**.

Given a normal logic program **BG**, an interpretation I, and an integrity constraint C, we say that I *satisfies* C *given a background knowledge* **BG**, or C *is true in* I *given* **BG**, if and only if $M(\mathbf{BG} \cup I) \models C$.

We say that a CLT T is *true in an interpretation* I *given* **BG** (or T satisfies I given **BG**, T covers I given **BG**, or I is positive given T and **BG**) if and only if I *satisfies each constraint* in T given **BG**.

If T is true in I ($I \models T$) we say that I is a model of T. If at least one constraint of the theory is false in an interpretation I, the whole theory T is false in I.

An IC is *range-restricted* if all the variables that occur in its head also occur in its body. As shown in [14], a range-restricted IC $L_1, \ldots, L_b \rightarrow A_1; \ldots; A_h$ is true in an interpretation I given a background knowledge **BG** if and only if the query

$$? - L_1, \ldots, L_b, \neg A_1, \ldots, \neg A_h. \tag{3}$$

fails against a Prolog database containing the atoms of I as facts together with the rules of the normal program **BG**. Note that if **BG** is range-restricted, every answer to a query Q against $\mathbf{BG} \cup I$ completely instantiates Q. Since, by definition, each variable in each query $\neg A_j$ occurs in a query L_i, $\neg A_j$ is ground when it is called.

Example 1 (from [1]) The Bongard Problems were introduced by the Russian scientist M. Bongard in his book [9]. Each problem consists of a number of pictures, some positive and some negative. The goal is to discriminate between the two classes. Each picture is composed of one or more figures, such as triangles, squares, circles, etc. Each figure has some properties, such as being small, large, pointing in a direction, etc. Moreover, relationships are defined between figures, such as inside, above, larger, and so on. Figure 1 shows some of these pictures.

Each picture can be defined by a set of atoms describing the properties and relationships of the figures in the pictures, i.e., an interpretation.

For instance, the left picture consists of a small triangle (identified by number 2) inside a small square (1) inside a large triangle (0).

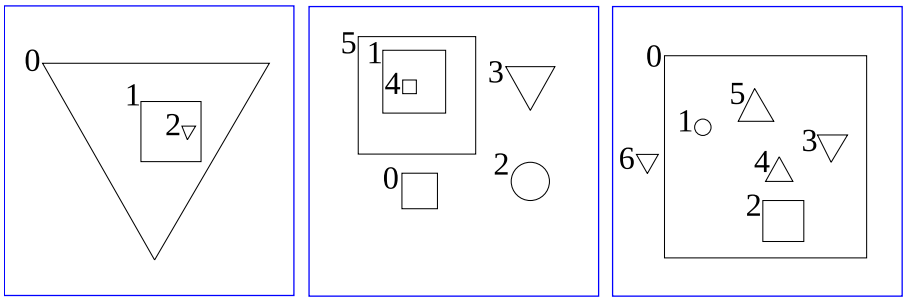


Fig. 1 Bongard pictures.

The interpretation that describes the picture is

$$I_l = \{triangle(0), large(0), square(1), small(1), inside(1,0), \\ triangle(2), small(2), inside(2,1)\} \quad (4)$$

Additional properties and relationships can be defined, for example by means of the following background knowledge **BG**:

$$in(A,B) \leftarrow inside(A,B). \quad (5)$$
$$in(A,D) \leftarrow inside(A,C), in(C,D). \quad (6)$$

Thus $M(\mathbf{BG} \cup I_l)$ will contain, besides all the atoms in I_l, the atoms $in(1,0)$, $in(2,1)$ and $in(2,0)$.

The IC

$$C = square(S), in(T,S) \rightarrow square(T) \quad (7)$$

requires any figure contained in a square to be a square itself. C is false in I_l given **BG** because, for the grounding substitution $\theta = \{S/1, T/2\}$, $Body(C)\theta$ is true and the only disjunct in $Head(C)\theta$ is false.

In the center picture instead C is true given **BG** because all the figures contained in squares are squares.

Learning from interpretations in ILP [15, 7] can be summarized as follows.
Given

- a set $\mathcal{I}^+ = \{I_1, \ldots, I_Q\}$ of positive interpretations (positive examples)
- a set $\mathcal{I}^- = \{I_{Q+1}, \ldots, I_R\}$ of negative interpretations (negative examples)
- a normal logic program **BG** (background knowledge)
- a hypothesis space $\mathcal{H}$

Find: an hypothesis $T \in \mathcal{H}$ such that

- for all $I^+ \in \mathcal{I}^+$, $M(\mathbf{BG} \cup I^+) \models T$
- for all $I^- \in \mathcal{I}^-$, $M(\mathbf{BG} \cup I^-) \not\models T$

Thus we look for a CLT that discriminates the positive from the negative interpretations.

ICL learns from interpretations using a covering loop on the negative examples, an approach that is dual to the covering loop of top-down ILP algorithms that learn from entailment. ICL starts from an empty theory and adds one IC at a time. After the addition of an IC, the set of negative examples that are ruled out by the IC is removed from $\mathcal{I}^-$. The covering loop ends when no more ICs can be generated or when $\mathcal{I}^-$ becomes empty (all the negative examples are ruled out). ICL is shown in Algorithm 1.

Algorithm 1 Function ICL

```
1: function ICL(I+, I−, BG, H, MaxBeamSize)
2:     T ← ∅                                                    ▷ Empty theory
3:     repeat
4:         C ← FindBestIC(I+, I−, BG, H, MaxBeamSize)
5:         if C ≠ ∅ then
6:             T ← T ∪ {C}
7:             Remove from I− all interpretations that are false for C
8:         end if
9:     until C = ∅ or I− is empty
10: return T
11: end function
```

The IC to be added in every iteration of the covering loop is returned by the procedure FindBestIC, shown in Algorithm 2. It uses a beam search

Algorithm 2 Function FindBestIC

```
1: function FindBestIC(I+, I−, BG, H, MaxBeamSize)
2:     Beam ← {true → false}
3:     BestIC ← ∅
4:     while Beam is not empty do
5:         NewBeam ← ∅
6:         for all C ∈ Beam do
7:             for all refinement Ref of C do
8:                 if Ref is better than BestIC then
9:                     BestIC ← Ref
10:                    if Ref is not to be pruned then
11:                        add Ref to NewBeam
12:                        if size of NewBeam > MaxBeamSize then
13:                            Remove worst clause from NewBeam
14:                        end if
15:                    end if
16:                end if
17:            end for
18:        end for
19:        Beam ← NewBeam
20:    end while
21:    return BestIC
22: end function
```

with $P(\ominus|\overline{C})$ as the heuristic function, where $P(\ominus|\overline{C})$ is the probability that an input example is negative given that it is ruled out by the IC C, i.e., it is the precision on negative examples. The search starts from the IC *true* $\rightarrow$ *false* that rules out all the negative examples but also all the positive examples and gradually refines that clause in order to make it more general. The maximum size of the beam is a user-defined parameter. The heuristic of each generated refinement is compared with the one of the best IC found so far and, if it is larger, the best IC is updated. At the end of the refinement cycle, the best IC found is returned.

The refinement operator exploits θ-subsumption for defining a generality relation among ICs: an IC C θ-subsumes an IC D, written $C \leq_\theta D$, if there exists a substitution θ such that $C\theta \subseteq D$ where C and D are seen as logical clauses (sets of literals). The generality relation for ICs is defined in terms of θ-subsumption as for learning from entailment but in the opposite direction: an IC D is more general than an IC C ($D \leq_g C$) if $C \leq_\theta D$. So *true* $\rightarrow$ *false* is the most specific constraint and the search in FINDBESTIC proceeds bottom up.

Refinements are obtained by using a refinement operator that adds a literal to the body or head of the IC or applies a substitution.

Extended IC language In the following, we recall the extended syntax and semantics for ICs from [28], which we use in this work. In the extended language, an integrity constraint (IC) is a formula C of the form

$$L_1, \ldots, L_b \rightarrow \exists(P_1); \ldots; \exists(P_n); \forall\neg(N_1); \ldots; \forall\neg(N_m) \tag{8}$$

where each L_i is a literal and each P_j and N_k is a conjunction of literals. $L_1, \ldots, L_b$ is called the *body* of C ($Body(C)$) and $\exists(P_1); \ldots; \exists(P_n); \forall\neg(N_1); \ldots; \forall\neg(N_m)$ is called the *head* of C ($Head(C)$). The semicolon here represents a disjunction.

We call each P_j a P *conjunction* and each N_k an N *conjunction*. We call each $\exists(P_j)$ a P *disjunct* and each $\forall\neg(N_k)$ an N *disjunct*.

The variables that occur in the body are quantified universally with scope the IC. The variables in the head that do not occur in the body are quantified existentially if they occur in a P disjunct and universally if they occur in a N disjunct, with scope the disjunct they occur in.

A P disjunct $\exists(P_j)$ is true in an interpretation I if and only if there exists a grounding substitution θ_P that makes $P_j\theta_P$ true. A N disjunct $\forall\neg(N_k)$ is true in an interpretation I if and only if for each grounding substitution θ_N $N_k\theta_N$ is false in I.

An IC C is *true in an interpretation* I ($I \models C$) if and only if, for each substitution θ such that each literal in $Body(C)\theta$ is ground and true in I, at least one disjunct in $Head(C)\theta$ is true in I.

Similarly to disjunctive clauses, the truth of an IC as in formula (8) in an interpretation $M(\mathbf{BG} \cup I)$ can be tested by running the query:

$$?- Body(C), not(P_1), \ldots not(P_n), N_1, \ldots, N_m \tag{9}$$

in a database containing the clauses of **BG** and atoms of I as facts. If the N conjunctions in the head share some variables, then the following query must be issued

$$? - Body(C), not(P_1), \ldots, not(P_n), not(not(N_1), \ldots, not(N_m)) \tag{10}$$

that ensures that the N conjunctions are tested separately without instantiating the variables. If the query finitely fails, the IC is true in the interpretation; if the query succeeds, the IC is false in the interpretation. Therefore, **BG** should be written so as to avoid infinite loops; for example, if **BG** is acyclic then the evaluation will terminate for a large class of queries [4].

The algorithm DPML [28] was proposed for learning these extended constraints. DPML modifies ICL by using a different refinement operator. Given an IC D, the set of refinements $\rho(D)$ of D is obtained by performing one of the following operations:

- adding a literal to the body;
- adding a disjunct to the head: the disjunct can be
  - a formula $\exists(d_1 \wedge \ldots \wedge d_k)$ where $\{d_1, \ldots, d_k\}$ is the set of literals allowed in a P disjunct,
  - a formula $\forall\neg(d)$ where d is a literal allowed in a N disjunct;
- removing a literal from a P disjunct in the head;
- adding a literal to a N disjunct in the head.

3 Probabilistic Inductive Constraint Logic

A Probabilistic Constraint Logic Theory (PCLT) is a set of probabilistic integrity constraints C_i of the form

$$p_i \ :: \ L_1, \ldots, L_b \rightarrow \exists(P_1); \ldots; \exists(P_n); \forall\neg(N_1); \ldots; \forall\neg(N_m) \tag{11}$$

Each constraint C_i is associated with a probability $p_i \in [0, 1]$ and a PCLT T is a set of probabilistic constraints $\{p_1 :: C_1, \ldots, p_n :: C_n\}$.

A PCLT T defines a probability distribution on ground constraint logic theories called possible theories in this way: for each grounding of the body of each IC, we include the IC in a possible theory with probability p_i and we assume all groundings to be independent. The probability is to be interpreted as the strength of the IC: a probability p_i means that the sum of the probabilities of the possible theories where a grounding of the constraint is present is p_i.

The notion of possible theory is similar to notion of world in ProbLog [16] where a world is a normal logic program. However, in [16] the term world is used to denote both logic programs and (least) Herbrand models in literature. In general, the use of the term world is ambiguous and there is no standard *de facto* for the terminology to be used. For example, Sato rarely uses the word

"world" [42]. For these reasons, to avoid using ambiguous terminology we use the expression "possible theories".

Let us assume that constraint C_i has n_i substitutions $\theta_{i_1}, ..., \theta_{i_{n_i}}$ that ground its body. Let $C_{i1}, \ldots, C_{in_i}$ be the clauses $C_i\theta_{i_1}, \ldots, C_i\theta_{i_{n_i}}$ and let us call the ICs C_{ij} *instantiations* of C_i, i.e., its partial groundings. Thus, the probability of a possible theory w is given by the product:

$$P(W = w) = \prod_{i=1}^{n} \prod_{C_{ij} \in w} p_i \prod_{C_{ij} \notin w} (1 - p_i). \tag{12}$$

$P(W = w)$ so defined is a probability distribution over the set of possible theories W. In the following, we will indicate $P(W = w)$ simply as $P(w)$.

The probability $P(\oplus|w, I)$ of the positive class given an interpretation I, a background knowledge **BG** and a possible theory w is defined as the probability that w satisfies I given **BG**[1]. Of course, its value is $P(\oplus|w, I) = 1$ if $M(\mathbf{BG} \cup I) \models w$ and 0 otherwise. The probability $P(\oplus|I)$ of the positive class given an interpretation I and a background **BG** is the probability of a PCLT T satisfying I given **BG**. From now on we always assume **BG** as given and we do not mention it again. $P(\oplus|I)$ is given by

$$P(\oplus|I) = \sum_{w \in W} P(\oplus, w|I) = \sum_{w \in W} P(\oplus|w, I)P(w|I) = \sum_{w \in W, M(\mathbf{BG} \cup I) \models w} P(w) \tag{13}$$

The probability $P(\ominus|I)$ of the negative class given an interpretation I is the probability of I not satisfying T and is given by $1 - P(\oplus|I)$.

Computing $P(\oplus|I)$ with Formula (13) is impractical as there is an exponential number of possible theories. We can associate a Boolean random variable X_{ij} to each instantiated constraint C_{ij} with the meaning that $X_{ij} = 1$ in a possible theory if C_{ij} is included in the possible theory. As p_i is associated with C_i, $P(X_{ij}) = p_i$ and $P(\overline{X_{ij}}) = 1 - p_i$. Let $\mathbf{X}$ be the set of the X_{ij} variables. These variables are all mutually independent. A valuation ν is an assignment of a truth value to all variables in $\mathbf{X}$. There is clearly a one to one correspondence between possible theories and valuations. A valuation can be represented as a set containing X_{ij} or $\overline{X_{ij}}$ for each X_{ij} and corresponds to the formula ϕ_ν obtained by conjoining all the X_{ij} variables:

$$\phi_\nu = \bigwedge_{i=1}^{n} \bigwedge_{X_{ij} \in \nu} X_{ij} \bigwedge_{\overline{X_{ij}} \in \nu} \overline{X_{ij}}. \tag{14}$$

Suppose a ground IC C_{ij} is violated in I. The possible theories where X_{ij} holds in the respective valuation are thus excluded from the summation in Formula (13). We must keep only the possible theories where $\overline{X_{ij}}$ holds in the

[1] **BG** is omitted from the formula for the sake of brevity.

respective valuation for all ground constraints C_{ij} violated in I. So I satisfies all the possible theories where the formula

$$\phi = \bigwedge_{i=1}^{n} \bigwedge_{M(\mathbf{BG}\cup I)\not\models C_{ij}} \overline{X_{ij}} \tag{15}$$

is true in the respective valuations, so

$$P(\oplus|I) = P(\phi) = \prod_{i=1}^{n} (1-p_i)^{m_i} \tag{16}$$

where m_i is the number of groundings of C_i that are not satisfied in I, since the random variables are all mutually independent. Since computing a^b is $O(\log b)$ with the "square and multiply" algorithm [23], $P(\oplus|I)$ can be computed in a time that is $O(n \log m)$ where m is the maximum number of groundings of constraints that are violated. Example 2 shows the application of Eq. 16 to the Bongard Problems domain. Example 3 shows how the computation of the probability has lower complexity than that required, for instance, by LPADs in a similar domain.

Example 2 (Example 1 continued) Consider the PCLT

$$\{C_1 = 0.5 \ :: \ triangle(T), square(S), in(T,S) \rightarrow false\} \tag{17}$$

In the left picture of Figure 1 the body of C_1 is true for the single substitution $T/2$ and $S/1$ thus $m_1 = 1$ and $P(\oplus|I_l) = 0.5$. In the right picture of Figure 1 the body of C_1 is true for three couples (triangle, square) thus $m_1 = 3$ and $P(\oplus|I_r) = 0.125$.

Example 3 Consider the following LPAD [45], inspired to the Bongard Problems:

$$\begin{aligned} class(pos) &: 0.3 \leftarrow triangle(T), square(S), in(T,S). \\ in(A,B) &: 0.3 \leftarrow inside(A,B). \\ in(A,D) &: 0.3 \leftarrow inside(A,C), in(C,D). \end{aligned}$$

plus an interpretation describing a picture. Notice that we have made the clauses for $in/2$ probabilistic. We can use this program to classify the picture, that is, we can ask the query $class(pos)$ and obtain its probability. To do so, inference algorithms find all explanations of the query atom and then make them mutually exclusive. Finding all explanations for the query means finding all the rule groundings that contribute to the truth of the query, and making the explanations mutually exclusive is a #P-hard problem. PCLTs do not have this problem because each constraint is independent of the others and we do not allow probabilities in the background knowledge.

Counting the number of groundings that are violated is a generalization of subsumption testing, which is NP-complete [24] in the length of both clauses to be tested for subsumption. However, since the length of the clauses we consider in learning is limited by an hyperparameter that is usually small enough, finding the number of groundings is not an issue.

3.1 Discussion of the Variable Independence Assumption

Considering the variables mutually independent may seem a strong restriction. However, in this section we will show that this is not a limitation and that under this assumption we can model every conditional probability distribution of the class variable given the atom variables, possibly by resorting to the addition of extra random variables.

Given a PCLT T containing a positive or negative class and a Herbrand base, we want to define a conditional probability distribution over a random variable C representing the class, given the value of the random variables $A_1, \ldots, A_n$ representing the Herbrand base.

In this way, the probability distribution represents the conditional dependence of the class given an interpretation, where the interpretation defines the value of the atoms of the Herbrand base, without modelling at the same time the dependence among atoms of the Herbrand base.

This is strictly related to the definition of discriminative models with conditional random fields [27] to model a relationship between class variables and input variables, rather than a relationship among input variables.

We can create a Bayesian network as shown in Figure 2 defined by the PCLT T. In this Bayesian network the variables associated with ground atoms are all parents of the class variable[2]. For example, suppose we want to model a general conditional dependence between the class atom and a Herbrand base containing two atoms: a and b. This dependence can be represented with the Bayesian network of Figure 3, where the conditional probability table (CPT) has four parameters, $p_1, \ldots, p_4$, so it is the most general. Let us call P' the distribution defined by this network.

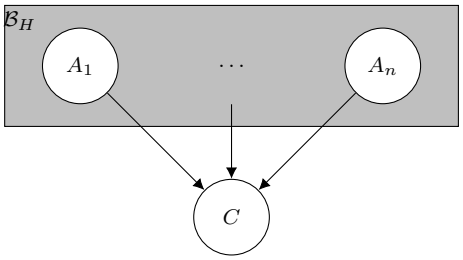
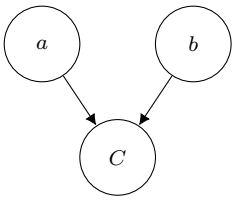


Fig. 2 Bayesian Network representing the dependence between the class of an interpretation and the Herbrand base $\mathcal{B}_H$.

[2] Which differs from a naive Bayes model because there the input variables (ground atoms) are all children of the class variable. This is a significant difference because the model in Figure 2 can have up to 2^n parameters if n is the number of ground atoms.



$P'(C\|a,b)$		C	
a	b	$\ominus$	$\oplus$
0	0	$1-p_1$	p_1
0	1	$1-p_2$	p_2
1	0	$1-p_3$	p_3
1	1	$1-p_4$	p_4

Fig. 3 Bayesian Network representing the dependence between class C and atoms a, b.

This model can be represented with the following PCLT

$$C_1 = 1 - p_1 :: \neg a, \neg b \rightarrow \mathit{false} \tag{18}$$
$$C_2 = 1 - p_2 :: \neg a, b \rightarrow \mathit{false} \tag{19}$$
$$C_3 = 1 - p_3 :: a, \neg b \rightarrow \mathit{false} \tag{20}$$
$$C_4 = 1 - p_4 :: a, b \rightarrow \mathit{false} \tag{21}$$

If we consider the interpretation $\{\}$ assigning value false to each atom of the Herbrand base, only constraint C_1 is violated. Thus, the probability that the class variable assumes value $\oplus$ is

$$P(C = \oplus | \neg a, \neg b) = 1 - (1 - p_1) = p_1 = P'(C = \oplus | \neg a, \neg b) \tag{22}$$

If we consider the opposite interpretation $\{a, b\}$, only constraint C_4 is violated and the probability of the positive class $P(C = \oplus | a, b) = p_4$ is equivalent to the probability assigned by the Bayesian network $P'(C = \oplus | a, b)$. It is easy to see that this holds also for the other possible interpretations, proving that the probability assigned to the positive class by the above PCLT always coincides with the one assigned by the Bayesian network of Figure 3.

Using the above PCLT is equivalent to representing the Bayesian network of Figure 3 with the Bayesian network of Figure 4, where a Boolean variable X_i represents whether constraint C_i is included in the possible theory (i.e., if it is enforced) and a Boolean variable Y_i whether constraint C_i is violated. Let us call P'' the distribution defined by this network. The conditional probability tables for nodes X_is are $P''(X_i = 1) = 1 - p_i$, those for nodes Y_is encode the deterministic functions

$$Y_1 = X_1 \wedge \neg a \wedge \neg b \tag{23}$$
$$Y_2 = X_2 \wedge \neg a \wedge b \tag{24}$$
$$Y_3 = X_3 \wedge a \wedge \neg b \tag{25}$$
$$Y_4 = X_4 \wedge a \wedge b \tag{26}$$

and that for C encodes the deterministic function

$$C = \neg Y_1 \wedge \neg Y_2 \wedge \neg Y_3 \wedge \neg Y_4 \tag{27}$$

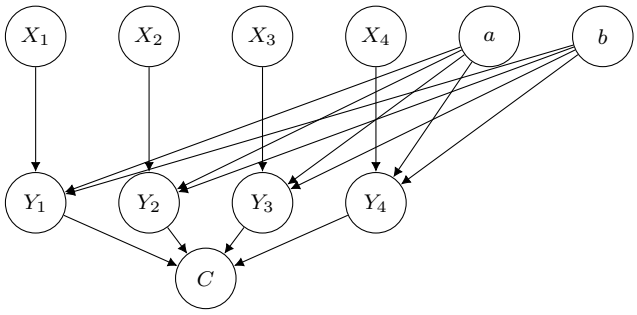


Fig. 4 Bayesian Network modeling the distribution P'' over C, a, b, $X_1, \ldots X_4$, $Y_1, \ldots Y_4$.

where C is interpreted as a Boolean variable with 1 corresponding to $\oplus$ and 0 to $\ominus$. If we want to compute $P''(C|\neg a, \neg b)$ we get

$$\begin{aligned}
P''(C|\neg a \neg b) &= \sum_{\mathbf{Y},\mathbf{X}} P''(X_1)\ldots P''(X_4)P''(Y_1|X_1,\neg a,\neg b)\ldots P''(Y_4|X_4,\neg a,\neg b) \\
&\quad P''(C|Y_1,Y_2,Y_3,Y_4) \\
&= p_1 \sum_{X_2,X_3,X_4,Y_2,Y_3,Y_4} P''(X_2)\ldots P''(X_4)P''(C|Y_1=0,Y_2,Y_3,Y_4) \\
&\quad P''(Y_2|X_2,\neg a,\neg b)\ldots P''(Y_4|X_4,\neg a,\neg b) \\
&= p_1 \sum_{X_2,X_3,X_4} P''(X_2)\ldots P''(X_4)P''(C|Y_1=0,Y_2=0,Y_3=0, \\
&\quad Y_4=0)P''(Y_2=0|X_2,\neg a,\neg b)\ldots P''(Y_4=0|X_4,\neg a,\neg b) \\
&= p_1 \sum_{X_2,X_3,X_4} P''(X_2)\ldots P''(X_4) \\
&= p_1 \qquad (28)
\end{aligned}$$

where $\mathbf{X} = \{X_1, \ldots, X_4\}$ and $\mathbf{Y} = \{Y_1, \ldots, Y_4\}$. Similarly, it is possible to show that P and P'' coincide for the other possible interpretations. If we look at the network in Figure 4 we see that the $\mathbf{X}$ variables are mutually unconditionally independent, showing that it is possible to represent any conditional dependence of C from the Herbrand base by using independent random variables. Of course, not assuming independence may result in a finer modeling of the domain. However, this would preclude PCLTs' nice computational properties. Achieving tractability requires approximations and we think that constraint independence is a reasonable assumption, similar to the independence among probabilistic choices in the distribution semantics for PLP.

Moreover, PCLTs can compactly encode the dependence because they can take advantage of context specific independence [37]. For example, in the CPT in Table 1 the probability of $C = \oplus$ does not depend on b when a is true. This

$P'(C\|a,b)$		C	
a	b	$\ominus$	$\oplus$
0	0	$1-p_1$	p_1
0	1	$1-p_2$	p_2
1	0	$1-p_3$	p_3
1	1	$1-p_3$	p_3

Table 1 A CPT with context specific independence.

dependence can be encoded with

$$C_1 = 1 - p_1 :: \neg a, \neg b \rightarrow \mathit{false} \tag{29}$$
$$C_2 = 1 - p_2 :: \neg a, b \rightarrow \mathit{false} \tag{30}$$
$$C_3 = 1 - p_3 :: a \rightarrow \mathit{false} \tag{31}$$

4 Learning the Parameters of Probabilistic Constraint Logic Theories

Let us consider first the parameter learning problem that can be expressed as follows.
Given

- a PCLT theory T
- a set $\mathcal{I}^+ = \{I_1, \ldots, I_Q\}$ of positive interpretations
- a set $\mathcal{I}^- = \{I_{Q+1}, \ldots, I_R\}$ of negative interpretations
- a normal logic program **BG**

Find: the parameters of T such that the likelihood

$$L = \prod_{q=1}^{Q} P(\oplus|I_q) \prod_{r=Q+1}^{R} P(\ominus|I_r) \tag{32}$$

is maximized. The likelihood is given by the probability that the example labels are observed for each example.

The likelihood can be unfolded to

$$L = \prod_{q=1}^{Q} \prod_{l=1}^{n} (1-p_l)^{m_{lq}} \prod_{r=Q+1}^{R} \left(1 - \prod_{l=1}^{n} (1-p_l)^{m_{lr}}\right) \tag{33}$$

where m_{lq} (m_{lr}) is the number of instantiations of C_l that are false in I_q (I_r) and n is the number of ICs. Let us compute the derivative of the likelihood with respect to the parameter p_i. We first aggregate the positive examples

$$L = \prod_{l=1}^{n} (1-p_l)^{m_{l+}} \prod_{r=Q+1}^{R} \left(1 - \prod_{l=1}^{n} (1-p_l)^{m_{lr}}\right) \tag{34}$$

where $m_{l+} = \sum_{q=1}^{Q} m_{lq}$. Then the partial derivative with respect to p_i is

$$\begin{aligned}
\frac{\partial L}{\partial p_i} &= \frac{\partial \prod_{l=1}^{n}(1-p_l)^{m_{l+}}}{\partial p_i} \prod_{r=Q+1}^{R}\left(1-\prod_{l=1}^{n}(1-p_l)^{m_{lr}}\right) \\
&\quad + \prod_{l=1}^{n}(1-p_l)^{m_{l+}} \frac{\partial \prod_{r=Q+1}^{R}\left(1-\prod_{l=1}^{n}(1-p_l)^{m_{lr}}\right)}{\partial p_i} \\
&= -m_{i+}(1-p_i)^{m_{i+}-1} \prod_{l=1,l\neq i}^{n}(1-p_l)^{m_{l+}} \prod_{r=Q+1}^{R}\left(1-\prod_{l=1}^{n}(1-p_l)^{m_{lr}}\right) \\
&\quad + \prod_{l=1}^{n}(1-p_l)^{m_{l+}} \sum_{r=Q+1}^{R} m_{ir}(1-p_i)^{m_{ir}-1} \prod_{l=1,l\neq i}^{n}(1-p_l)^{m_{lr}} \\
&\quad \cdot \prod_{r'=Q+1,r'\neq r}^{R}\left(1-\prod_{l=1}^{n}(1-p_l)^{m_{lr'}}\right) \\
&= -m_{i+}(1-p_i)^{m_{i+}-1} \prod_{l=1,l\neq i}^{n}(1-p_l)^{m_{l+}} \frac{(1-p_i)^{m_{i+}}}{(1-p_i)^{m_{i+}}} \\
&\quad \prod_{r=Q+1}^{R}\left(1-\prod_{l=1}^{n}(1-p_l)^{m_{lr}}\right) + \\
&\quad \prod_{l=1}^{n}(1-p_l)^{m_{l+}} \sum_{r=Q+1}^{R} m_{ir} \frac{\prod_{l=1}^{n}(1-p_l)^{m_{lr}}}{1-p_i} \\
&\quad \prod_{r'=Q+1,r'\neq r}^{R}\left(1-\prod_{l=1}^{n}(1-p_l)^{m_{lr'}}\right) \cdot \frac{1-\prod_{l=1}^{n}(1-p_l)^{m_{lr}}}{1-\prod_{l=1}^{n}(1-p_l)^{m_{lr}}} \\
&= -\frac{m_{i+}(1-p_i)^{m_{i+}-1}L}{(1-p_i)^{m_i}} + \sum_{r=Q+1}^{R} \frac{m_{ir}\prod_{l=1}^{n}(1-p_l)^{m_{lr}}L}{(1-p_i)(1-\prod_{l=1}^{n}(1-p_l)^{m_{lr}})} \\
&= -\frac{m_{i+}L}{1-p_i} + \sum_{r=Q+1}^{R} \frac{m_{ir}\prod_{l=1}^{n}(1-p_l)^{m_{lr}}L}{(1-p_i)(1-\prod_{l=1}^{n}(1-p_l)^{m_{lr}})} \\
&= \frac{L}{1-p_i}\left(\sum_{r=Q+1}^{R} \frac{m_{ir}\prod_{l=1}^{n}(1-p_l)^{m_{lr}}}{1-\prod_{l=1}^{n}(1-p_l)^{m_{lr}}} - m_{i+}\right) \\
&= \frac{L}{1-p_i}\left(\sum_{r=Q+1}^{R} m_{ir}\frac{P(\oplus|I_r)}{P(\ominus|I_r)} - m_{i+}\right) \qquad (35)
\end{aligned}$$

The equation $\frac{\partial L}{\partial p_i} = 0$ does not admit a closed form solution so we must use optimization to find the maximum of L.

We can optimize the likelihood with gradient descent [11], where weights are updated using the formula

$$\mathbf{p}_{n+1} = \mathbf{p}_n - \epsilon \nabla_{\mathbf{p}} L(\mathbf{p}) = \mathbf{p}_n - \epsilon \nabla_{\mathbf{p}} \frac{\partial L}{\partial \mathbf{p}} \qquad (36)$$

where ϵ is the learning rate defining the size of the step done by gradient descent along the gradient and $\mathbf{p}$ is the vector containing the parameters p_i, or with a second order method such as Limited-memory BFGS (L-BFGS) [34].

In the experiments we report results only for gradient descent, as it outperforms L-BFGS in most cases in terms of area under the PR and ROC curves, and execution time.

5 Learning the Structure of Probabilistic Constraint Logic Theories

The structure learning problem can be expressed as
Given

- a set $\mathcal{I}^+ = \{I_1, \ldots, I_Q\}$ of positive interpretations
- a set $\mathcal{I}^- = \{I_{Q+1}, \ldots, I_R\}$ of negative interpretations
- a normal logic program **BG**
- a language bias

Find: a PCLT T such that the likelihood

$$L = \prod_{q=1}^{Q} P(\oplus|I_q) \prod_{r=Q+1}^{R} P(\ominus|I_r) \tag{37}$$

is maximized.

The PASCAL algorithm solves this problem by first identifying good candidate ICs and then searching for a theory guided by the log likelihood (LL) of the data.

5.1 The PASCAL Algorithm

PASCAL is shown in Algorithm 3. It takes as input the positive and negative interpretations and a list of settings defining the hypothesis space. It returns a theory T of probabilistic ICs. After the search in the space of ICs, encoded in lines 2 - 19, PASCAL performs a greedy search in the space of theories, described in lines 20 - 28.

Thanks to the last part of the algorithm, based on a greedy search, PASCAL uses a search bias that should work against overfitting. This is confirmed by Section 7, where the experimental results computed through cross-validation are comparable with those of other systems.

Language bias The search over the space of constraints to identify the candidate ones is performed according to a language bias expressed by means of *mode* declarations. Following [30], a mode declaration m is either a head declaration $modeh(r, s)$ or a body declaration $modeb(r, s)$, where s, the *schema*, is a ground literal, and r is an integer called the *recall*. A schema is a template for literals in the head or body of a constraint and can contain special

Algorithm 3 Function PASCAL

```
1: function PASCAL(I+, I−, BG, NC, MLB, MD, MLP, MLN, BeamSize, MaxSteps)
2:     Steps = 1
3:     Beam ← (false ← true, −∞)                                   ▷ Empty IC
4:     repeat
5:         NewBeam = []
6:         while Beam is not empty do                               ▷ ICs search
7:             Remove the first IC (C, LL) from Beam
8:             Ref ← all refinements of C respecting MLB,MD,MLP, and MLN
9:             for all C′ ∈ Ref do
10:                ({C″}, LL″) ←LearnParams({C′}, I+, I−, BG)       ▷ gradient descent
11:                NewBeam ←Insert((C″, LL″), NewBeam)
12:                if size(NewBeam) > BeamSize then
13:                    Remove the last element of NewBeam
14:                end if
15:            end for
16:        end while
17:        Beam ← NewBeam
18:        Steps = Steps + 1
19:    until Steps > MaxSteps
20:    T ← ∅, LL ← −∞                                               ▷ Theory search
21:    repeat
22:        Remove the first couple (C, LL) from Beam
23:        (T′, LL′) ←LearnParams(T ∪ {C}, I+, I−, BG)
24:        if LL′ > LL then
25:            T ← T′, LL ← LL′
26:        end if
27:    until Beam is empty or T contains NC ICs
28:    return T
29: end function
```

placemark terms of the form `#type`, `+type` and `-type`, which stand, respectively, for ground terms, input variables and output variables of a type. An input variable in a body literal of a constraint must be an output variable in a preceding body literal in the IC. Similarly, an input variable in the head must be either an output variable in a preceding literal in the same disjunct or in a body literal in the IC. If M is a set of mode declarations, $L(M)$ is the *language of* M, i.e. the set of ICs $A_1; \ldots; A_n :- L_1, \ldots, L_b$ such that the head atoms A_i (resp. body literals L_i) are obtained from some head (resp. body) declaration in M by replacing all `#` placemarks with ground terms and all `+` (resp. `-`) placemarks with input (resp. output) variables.

ICs Search The first phase aims at searching the space of constraints for a set of promising ones in terms of log-likelihood (LL). In this step, a beam search is performed: initially the beam contains only the empty clause $true \rightarrow false$ with score $LL_0 = -\infty$. Then, PASCAL enters the refinement cycle (Alg. 3, lines 6–16) in order to output a list of at most *BeamSize* candidate ICs sorted by decreasing LL. *BeamSize* is a user-defined setting storing the maximum size of the beam.

For each IC C, refinements are generated by means of the operators described in Section 2 where the literals allowed in the body and in the head

are defined by the mode declarations. Moreover, the user can set the following bounds:

- MLB, the maximum number of literals in the body of ICs;
- MD, the maximum number of disjuncts in the head of ICs;
- MLP and MLN, the maximum number of literals allowed in a P disjunct and a N disjunct respectively.

In line 10 of Algorithm 3 parameter learning is executed on a theory composed of the single refined clause - $\{C'\}$ - by function *LearnParams*, employing either gradient descent or L-BFGS. The initial values for the parameters are randomly set. The resulting log likelihood LL'' is used as the score of the updated IC C''. The scored refinements are inserted back into the beam in order of decreasing score. If the beam exceeds the maximum size $BeamSize$, the last element is removed. Function *Insert* at line 11 is used to update the beam.

Beam search is repeated until the beam becomes empty or a maximum number of *Steps* is reached.

Theory search The second phase is a greedy search in the space of theories starting with an empty theory T with the lowest value of LL (line 20). Then one IC at a time is added from the *Beam*. After each addition, parameter learning is run on the extended theory $T \cup C$ and the log likelihood LL' of the data is computed as the score of the resulting theory T'. If LL' is better than the current best, the IC is kept in the theory, otherwise it is discarded. This is done for each clause in *Beam*, until the *Beam* is empty or a maximum number NC of ICs, defined by the user, is reached. In line 28 a PCLT T is returned.

5.2 Execution Example

We now show an example of execution for the BUPA dataset that is used later in the experiments. BUPA[3] is a medical dataset for diagnosing liver disorders. The dataset uses 9 predicates, `alkphos/2`, `gammagt/2`, `mcv/2`, `sgot/2`, `sgpt/2`, `drinks/2`, `bupa_name/1`, `bupa_type/1`, `bupa/2`. Each interpretation records a list of ground facts representing values for blood tests (the first five predicates) and the number of half-pints drunk per day (the sixth) for a single male individual. The last three predicates were artificially created by researchers who defined the BUPA dataset in order to split data into train and test sets and give a target predicate, `bupa/2`, to learn. Positive interpretations represent individuals who have liver disorders.

[3] `https://relational.fit.cvut.cz/dataset/Bupa`

An example of positive interpretation is:

$$
\begin{aligned}
&alkphos(1, t1, 92).\\
&mcv(1, t1, 85).\\
&gammagt(1, t1, 31).\\
&drinks(1, t1, 0.000).\\
&sgot(1, t1, 27).\\
&sgpt(1, t1, 45).\\
&bupa_k(1, pos).
\end{aligned}
$$

For a negative interpretation, 'pos' in `bupa_k` is replaced with 'neg'. Note that the first argument of each fact is the interpretation's ID (also called key), which is not a descriptive argument of the individual represented by the interpretation. For this reason, it has not been counted in the predicates' arity listed above.
The language bias used is

$$
\begin{aligned}
&modeh(1, alkphos(+arg1, -alkv)).\\
&modeh(1, drinks(+arg1, -drinkv)).\\
&modeh(1, gammagt(+arg1, -gammav)).\\
&modeh(1, mcv(+arg1, -mcvv)).\\
&modeh(1, sgpt(+arg1, -sgptv)).\\
&modeh(1, sgot(+arg1, -sgotv)).
\end{aligned}
$$

$$
\begin{aligned}
&modeb(1, alkphos(-arg1, -alkv)).\\
&modeb(1, drinks(-arg1, -drinkv)).\\
&modeb(1, gammagt(-arg1, -gammav)).\\
&modeb(1, mcv(-arg1, -mcvv)).\\
&modeb(1, sgpt(-arg1, -sgptv)).\\
&modeb(1, sgot(-arg1, -sgotv)).
\end{aligned}
$$

and the algorithm settings take the following values: *BeamSize=2, MLB=2, MD=2, MLP=MLN=1, NC=8.*

When searching the *space of ICs*, the starting IC $true \rightarrow false$ is extracted from the initial $Beam$ and is refined using the $modeb$ declarations. $Modeh$ declarations are considered but do not produce revisions as they all have an input argument; since the body is empty, no variable can be placed in the input argument. This leads to the following clauses in the *first beam cycle*:

1. First refinement: C'

$$0.5 :: alkphos(A, B) \rightarrow false.$$

After gradient descent optimization: C''

$$0.360146 :: alkphos(A, B) \rightarrow false.$$

with $LL = -216.919$. The couple $(C'', -216.919)$ is inserted in the beam.

2. Second refinement: C'

$$0.5 :: drinks(A, B) \rightarrow false.$$

After gradient descent optimization: C''

$$0.356961 :: drinks(A, B) \rightarrow false.$$

with $LL = -217.806$. The couple $(C'', -217.806)$ is inserted in the beam.

Six refinements are generated (one for each *modeb*), the best being (clause with the largest $LL = -214.592$)

$$0.368780 :: gammagt(A, B) \rightarrow false.$$

In the *second beam cycle*, C:

$$gammagt(A, B) \rightarrow false$$

is extracted (being at the top of the beam) and its body is refined based on all modeb declarations, as $MLB = 2$, leading to:

1. First refinement: C'

$$0.5 :: gammagt(A, B), alkphos(C, D) \rightarrow false.$$

After gradient descent optimization: C''

$$0.372082 :: gammagt(A, B), alkphos(C, D) \rightarrow false.$$

with $LL = -213.732$. The couple $(C'', -213.732)$ is inserted in the beam.

2. Second refinement: according to the placemark terms in the *modeb* declarations, also the refinement C':

$$0.5 :: gammagt(A, B), alkphos(A, C) \rightarrow false.$$

can be generated and optimized.

3. These two types of refinements are repeated by generating and optimizing all rules of the form:

$$0.5 :: gammagt(A, B), L_2 \rightarrow false.$$

with L_2 being every *modeb* literal except for $gammagt/2$.

4. Then, all refinements of the form:

$$0.5 :: gammagt(A, B) \rightarrow \exists(P_1).$$
$$0.5 :: gammagt(A, B) \rightarrow \forall\neg(N_1).$$

are generated and optimized, with P_1 and N_1 disjuncts containing one of each *modeh* atom at a time (as $MLP = MLN = 1$). This is possible because the input arguments of the *modeh* declarations can be replaced with variables appearing in the body.
5. After having built all possible refinements based on the literal $gammagt/2$, the current best theory ($LL = -213.732$) is still
$0.372082 :: gammagt(A, B), alkphos(C, D) \rightarrow false.$

Now, C:

$$0.5 :: sgot(A, B) \rightarrow false.$$

is extracted, being the second clause in the beam with $LL = -215.119$, and the previous 1-4 steps are repeated using $sgot/2$.
This is the end of the second beam cycle, terminating with the best theory found among all refinements:
$0.372082 :: gammagt(A, B), alkphos(C, D) \rightarrow false$, which is therefore put at the top of the beam.

In the *third beam cycle, C*:

$$0.372082 :: gammagt(A, B), alkphos(C, D) \rightarrow false.$$

is extracted and all refinements of the form:

$$0.5 :: gammagt(A, B), alkphos(C, D) \rightarrow \exists(P_1).$$
$$0.5 :: gammagt(A, B), alkphos(C, D) \rightarrow \exists(P_1); \exists(P_2).$$
$$0.5 :: gammagt(A, B), alkphos(C, D) \rightarrow \forall\neg(N_1).$$
$$0.5 :: gammagt(A, B), alkphos(C, D) \rightarrow \forall\neg(N_1); \forall\neg(N_2).$$
$$0.5 :: gammagt(A, B), alkphos(C, D) \rightarrow \exists(P_1); \forall\neg(N_1).$$

are generated and optimized, with P_i and N_i containing one of each *modeh* atom at a time. Two disjuncts can be present in the head as $MD = 2$; given that $MLB = 2$, no further literal can be added to the body. After having built all possible refinements, the current best theory ($LL = -213.732$) is still
$0.372082 :: gammagt(A, B), alkphos(C, D) \rightarrow false.$

Now, C:

$$0.5 :: gammagt(A, C), alkphos(D, E) \rightarrow \forall\neg(sgot(A, B)). \quad (38)$$

is extracted, being the second clause in the beam with $LL = -213.885$, and it is refined by applying the operations listed at the end of subsection 2.2.

Beam cycles go on until the beam becomes empty and the best IC found so far is kept in *Beam* (line 17 of Alg.3), in this case
$0.372082 :: gammagt(A, B), alkphos(C, D) \rightarrow false.$

When the repeat-until cycle ends, *Beam* has kept the best ICs for the theory search.
When searching the *space of theories*, PASCAL generates the theory:

$0.178997 :: mcv(A, C), drinks(D, B) \rightarrow \forall\neg(drinks(A, B)); \forall\neg(mcv(A, C)).$
$0.179310 :: gammagt(A, C), drinks(A, D) \rightarrow \forall\neg(sgot(A, B)).$
$0.179607 :: sgpt(A, B), drinks(A, C) \rightarrow false.$
$0.180221 :: gammagt(A, B), alkphos(C, D) \rightarrow false.$

with a log likelihood of -187.826. Note that the last clause was the first best IC found.

6 Related Work

The approach for assigning a semantics to PCLTs is inspired by the distribution semantics [42]: a probabilistic theory defines a distribution over non-probabilistic theories by assuming independence among the choices in probabilistic constructs. The distribution semantics has emerged as one of the most successful approaches in Probabilistic Logic Programming (PLP) and underlies many languages such as Probabilistic Horn Abduction [35], Independent Choice Logic [36], PRISM [43], Logic Programs with Annotated Disjunctions [45] and ProbLog [16].

According to the distribution semantics, probabilistic inference aims at computing the probability that a ground atom is true. However, performing such task requires an expensive procedure that is usually based on knowledge compilation. For example, ProbLog [16] and PITA [40,41] build a Boolean formula and compile it into a language from which the computation of the probability is linear in the size of the resulting formula. However, the compilation procedure is #P in the number of random variables. On the contrary, computing the probability of the positive class given an interpretation in a PCLT is $O(n \log m)$, where n is the number of clauses and m is the maximum number of groundings, and computing m is polynomial in the database size.

PASCAL is related to the systems SLIPCASE [5] and SLIPCOVER [6] that learn probabilistic logic programs under the distribution semantics. However, these perform classification of target atoms rather than of interpretations. Recently, the LIFTCOVER algorithm [33] was proposed to perform discriminative learning of probabilistic logic programs that are limited to one layer of rules combined with noisy–or. PCLTs are the dual of this PLP formalism as ICL is the dual of the learning from entailment setting in ILP. In fact in [33] a single firing rule is enough to make the query true with a nonzero probability, while in PCLTs one single violated constraint is enough to make the class negative with a nonzero probability. The higher the number of firing rules (violated constraints), the higher is the probability of the positive (negative) class.

PCLTs can be related to Markov Logic Networks [39], as they share the capability to encode constraints on possible interpretations. The difference between them relies in the fact that MLNs can either encode a joint distribution over all atoms with a generative approach, or encode conditional probability distributions with a discriminative approach [44]: in the latter case their aim is to predict some query atom variables given the others. PASCAL performs discriminative learning too, but its aim is to classify interpretations, that is, encoding the probability distribution of the class variable given the atom variables. Given a PCLT, it is possible to obtain an MLN encoding the same distribution over the class variable given the values of all the atoms.
For example, the PCLT (18)-(21) can be emulated with the following MLN:

$$\ln(1-p_1)\ \ \neg a \wedge \neg b \wedge \neg C \tag{39}$$
$$\ln(p_1)\ \ \neg a \wedge \neg b \wedge C \tag{40}$$
$$\ln(1-p_2)\ \ \neg a \wedge b \wedge \neg C \tag{41}$$
$$\ln(p_2)\ \ \neg a \wedge b \wedge C \tag{42}$$
$$\ln(1-p_3)\ \ a \wedge \neg b \wedge \neg C \tag{43}$$
$$\ln(p_3)\ \ a \wedge \neg b \wedge C \tag{44}$$
$$\ln(1-p_4)\ \ a \wedge b \wedge \neg C \tag{45}$$
$$\ln(p_4)\ \ a \wedge b \wedge C \tag{46}$$

where C is an atom representing the class. If we compute the conditional probability of C given an interpretation I, we get the same results of the PCLT. In fact, consider the empty interpretation and call P''' the distribution defined by the MLN. We get

$$\begin{aligned} P'''(C = \oplus | \neg a, \neg b) &= P'''(C = \oplus, \neg a, \neg b)/P'''(\neg a, \neg b) \\ &= \frac{\frac{e^{\ln(p_1)}}{Z}}{\frac{e^{\ln(1-p_1)}+e^{\ln p_1}}{Z}} = \frac{e^{\ln(p_1)}}{e^{\ln(1-p_1)} + e^{\ln p_1}} \qquad (47) \\ &= \frac{p_1}{1-p_1+p_1} = p_1 \qquad (48) \end{aligned}$$

where Z is the partition function. Similarly for the other interpretations. So PCLTs are a specialization of MLNs that, by focusing on a simpler problem, allow better performance of inference algorithms. In other words, it is only possible to encode PCLTs with MLNs but not viceversa.
Finally, as regards parameter learning, in [44] parameters are learned using gradient descent as in PASCAL and, to limit the complexity, authors approximate expected counts by considering only the map state of the query variables. Differently from them, we do not need to perform approximations when computing the gradient as the model was specifically designed with discriminative inference in mind.

Tractable Markov Logic (TML) [20] is a subset of Markov Logic, where inference is kept tractable by imposing restrictions on the language: in particular, only subclass/instance, subpart and relation rules and facts can be

expressed, and class hierarchies are required to be forests. Our work differs from TML in two respects. First, inference in TML computes the conditional probability of a query given a theory, while we compute the conditional probability of a class given an interpretation. Second, PCLT constraints are more general than the rules that can be expressed in TML.

PCLTs are also related to FOProbLog [10], an algorithm which defines a probability distribution on interpretations built using ground atoms from the Herbrand base. In a theory, probability values are associated with facts, which are used as activators of the formulae of the theory. To compute the probability of a class C it builds the set of total choices. Then, it extends the total choices by adding ground atoms from the Herbrand base, extensions that are called models. Note that some of these models may be inconsistent, but in this case their probability is 0, thus we concentrate only on consistent extensions. Once all the models are collected, the probability of a query Q is defined as an interval $[p_1, p_2]$, where p_1 is the sum of the probabilities of the models where the query can be proved. On the other hand, $p_2 = 1 - p_{\neg Q}$, where $p_{\neg Q}$ is the probability of the query $\neg Q$ computed in the same way of the probability of Q. The probability interval for the query $\neg Q$ will be $[1 - p_1, p_{\neg Q}]$. So FOProbLog is similar to MLNs, it defines a probability distribution over interpretations or queries, while we define a probability distribution over the class only, thus tackling a simpler problem.

Another system related to PASCAL is 1BC [21], that induces first-order features in the form of conjunctions of literals and combines them using naive Bayes in order to classify examples. First-order features are similar to integrity constraints with an empty head: they check the existence of values for the variables that satisfy the conjunction. The probability of a feature is computed by relative frequency in 1BC. This can lead to suboptimal results if compared to PASCAL, where the probabilities are optimized to maximize the likelihood.

Another system which is close to our approach is TILDE [8] which applies First-Order Logical Decision Trees (FOLDT) to the problem of learning from interpretations. A FOLDT is a binary decision tree in which each node of the tree represents a conjunction of literals defined by the path to that node. Free variables in the literals can be shared across many nodes under the limitation that, starting from the first node that introduces the variable, all the other nodes must be in the left branch of their parent node. Such a limitation is due to the fact that each variable introduced is existentially quantified. The resulting learned clauses are used to classify interpretations given a set of possible classes. These clauses can be associated with the probability distribution of classes in the leaf that corresponds to the learned clause, therefore, TILDE can be used also to perform probabilistic classification. In this way, TILDE can act like relational probability trees [31], which build classification trees considering a larger feature space that also includes aggregation operators. However, both TILDE and relational probability trees can only return a probability value that is associated with the leaves of the constructed tree. Instead, PASCAL returns a wider range of values, because it considers also the number of satisfied groundings of the ICs during inference.

7 Experiments

We compared PASCAL with:

- the PLP algorithms LIFTCOVER [32], SLIPCOVER [6] and LEMUR [19];
- the MLNs algorithms BUSL [29], LSM [26], MLN-BC/MLN-BT [25];
- TILDE [8] as a representative of (probabilistic) relational classifiers.

We performed tests on the datasets of [32] plus the Bongard dataset [9], to which the Bongard Problem of Example 1 is inspired.

Note that SLIPCOVER, LIFTCOVER and LEMUR can be seen as a baseline for comparison with respect to PASCAL, since they have already been compared with many state-of-art systems in our previous works [6,19], demonstrating that they were competitive or superior with respect to the MLNs learning systems.

All PLP systems, included PASCAL, are implemented in SWI-Prolog [46].

Datasets Datasets are specific for the *learning from entailment* setting as they were used in [32] to compare LIFTCOVER with SLIPCOVER: they are composed of a set of mega-interpretations, each possibly containing more than one example (i.e., fact for a target predicate). However, those mega-interpretations contain in practice a single fact for the target predicate, so it is possible to classify each mega-interpretation as positive or negative depending on the target predicate example. For this reason, we could apply PASCAL by considering each mega-interpretation as an input interpretation.

Table 2 shows the datasets' features: number of different predicates, total number of tuples, number of positive and negative examples, and number of folds for cross-validation.

Table 2 Characteristics of the datasets for the experiments: number of predicates (Pred.), of tuples (i.e., ground atoms), of positive (I^+) and negative (I^-) examples, of folds.

Dataset	Pred.	Tuples	I^+	I^-	Folds
Financial	9	92658	34	223	10
Bupa	12	2781	145	200	5
Mondial	11	10985	572	616	5
Mutagenesis	20	15249	125	126	10
Sisyb	9	354507	3705	9229	10
Sisya	9	358839	10723	6544	10
Pyrimidine	29	2037	20	20	4
Yeast	12	53988	1299	5456	10
Triazine	62	10079	20	20	4
Carcinogenesis	36	24533	182	155	1
Bongard	5	2792	130	265	1

Algorithms' settings SLIPCOVER/LIFTCOVER/LEMUR and TILDE allow *modeh* and *predict* declarations only for the target predicate(s), respectively.

PASCAL, instead, by learning models that provide predictions at the level of interpretations, allows *modeh* declarations for all predicates of the domain (see subsection 5.2).

As described in Section 5.1, PASCAL offers the following settings: the size *BeamSize* of the beam, the maximum number of disjuncts *MD* per IC, the maximum number of literals *MLP* contained in positive disjuncts and the maximum number of literals *MLN* contained in negative disjuncts, the maximum number of body literals *MLB*, the maximum number *MaxSteps* of IC search iterations, and the maximum number of ICs *NC* that may be inserted into the final program. Table 3 summarizes the values taken by these settings. They were chosen with the objective of keeping the computation time below 24 hours per fold.

TILDE was executed with default values for its settings.

Table 3 Settings controlling PASCAL.

Dataset	*BeamSize*	*MLB*	*MD*	*MLP*	*MLN*	*NC*	Learning rate
Bupa	2	3	2	1	1	8	0.5
Carcinogenesis	1	3	2	1	1	7	0.5
Financial	1	2	1	1	1	8	0.05
Mondial	1	2	1	1	1	8	0.5
Mutagenesis	2	2	1	1	1	8	0.5
Pyrimidine	2	3	2	1	1	8	0.5
Sisya	1	2	1	1	1	4	0.05
Sisyb	1	2	0	0	0	8	0.05
Triazine	2	3	2	1	1	8	0.5
Yeast	2	2	2	1	1	4	0.5
Bongard	1	2	2	1	1	8	0.5

SLIPCOVER and LIFTCOVER settings can be found in Table 2 of [32] for all datasets except Bongard. We applied SLIPCOVER and LIFTCOVER on Bongard by setting $NB = 100, NI = 20, NInt = 4, NS = NA = 1, NV = 4, WMin = 0, NIS = 50$ (see Section 7 of [32]). LIFTCOVER can exploit either an Expectation Maximization (EM) algorithm [18] or L-BFGS [34] to maximize the log-likelihood during parameter learning, so results show both variants. LEMUR settings can be found in subsection 7.1 of [19] for the three datasets in common (Carcinogenesis, Mondial, Mutagenesis).

Results Experiments with PASCAL were performed on GNU/Linux machines with Intel Xeon E5-2697 v4 (Broadwell) at 2.30 GHz, using cross-validation.

TILDE can only be executed on 32-bit machines: we used a GNU/Linux machine with Intel Core 2 Quad CPU Q6600 at 2.40GHz and 3.6 GB of RAM, using cross validation. To compare the results of TILDE with the others we scaled its runtime of a factor 2.4/2.3.

LEMUR, MLN-BC and MLN-BT in [19] were executed on GNU/Linux machines with an Intel Core 2 Duo E6550 (2.333 GHz) processor: in order to

compare their runtime, we scaled PASCAL learning time of a factor 2300/2333 based on the different CPU clock speeds.

Results for all the other systems are referred to the same machines used for PASCAL.

For performance evaluation, we considered the Area Under the Precision Recall and ROC curves (AUCPR and AUCROC respectively) using the methods described in [13,38].

Tables 5, 6 and 7 show the AUC-PR, AUC-ROC and learning time in seconds respectively, averaged over the folds, for PASCAL, SLIPCOVER, LIFTCOVER and TILDE. SLIPCOVER and LIFTCOVER results are taken from Tables 3-5 of [32].

In Tables 8, 9 and 10 we report the AUC-PR, AUC-ROC and running time achieved by PASCAL, LEMUR, MLN-BC (with and without sampling) and MLN-BT (with and without sampling) on the datasets in common. LSM and BUSL were also considered, but they are not included in the tables because they were not able to complete the task due to an out of memory error. LEMUR, MLN-BC, MLN-BT results are taken from Tables 3, 5, 7 of [19].

Table 11 shows the p-value of a paired two-tailed t-test of the difference in AUC-PR and AUC-ROC between PASCAL and TILDE on all datasets, except for Carcinogenesis/Bongard, where we did not apply cross-validation, and Sisyb, where we got the same AUC values over all folds with both algorithms. The p-value is not reported for the other systems as data were not available.

Discussion As regards the quality of the theories learnt, Tables 5, 6, 8, and 9 show that PASCAL achieves the best AUC-PR 3 times out of 11 and comparable AUC-PR in the other cases except for Bongard. LIFTCOVER-EM and PASCAL are the best algorithms according to AUC-ROC in 4 cases out of 11 (Sisyb is not counted); in the other cases PASCAL gets comparable AUC-ROC except for Bongard and Sisya. PLP algorithms always beat MLNs. By looking at the characteristics of the datasets, we can observe that the three datasets where PASCAL performs well - Triazine, Financial and Carcinogenesis - have a small number of examples but a large number of different predicates, possibly indicating that the expressive language bias is beneficial when the dataset is not very big but has a rich structure.

The quality of the models built by PASCAL is especially influenced by the language bias and the choice of the settings' values which define the search space of the candidate ICs; datasets' size impacts execution time and search space. The values reported in Table 3 generated the PCLTs described in Table 4, in terms of average size of the theories (number of learnt ICs), average size of the constraints (number of atoms), characteristics of the constraints (number of P and N disjuncts).

As regards learning time, LIFTCOVER-EM is the fastest system, as can be seen in Tables 7 and 10. Note that, in spite of the greater expressiveness allowed by PASCAL language bias (w.r.t. to the other PLP systems), that is responsible for a larger search space, PASCAL learning times are in line with

Table 4 Characteristics of the PCLTs generated by PASCAL for each dataset.

Dataset	Avg theory size	Avg ICs size	Avg No P dis.	Avg No N dis.
Bupa	8	3.5	1	0.5
Carcinogenesis	7	3	1	1
Financial	4.9	3	0.5	0.5
Mondial	7	2.5	0.5	0
Mutagenesis	8	2.5	0.5	0.5
Pyrimidine	8	3	0.5	1
Sisya	3.8	2.5	0.25	0.6
Sisyb	2.3	2.7	0	0
Triazine	8	3	0.75	1
Yeast	4	2.7	1.75	0
Bongard	8	3	1	0.5

all algorithms, and in 3 cases out of 11 it is even the second best. Also, with respect to MLNs learning time, PASCAL is always faster except Carcinogenesis (Table 10).

T-tests show that area differences between PASCAL and TILDE are statistically significant, with a confidence level of 0.05, in 7 out of 14 cases.

Table 5 Average AUC-PR. LIFT-EM and LIFT-LBFGS columns show the results for LIFTCOVER using respectively EM and L-BFGS for parameter learning. In bold the best results for each dataset.

Dataset	SLIPCOVER	LIFT-EM	LIFT-LBFGS	PASCAL	TILDE
Bupa	**1**	**1**	**1**	**1**	0.420
Carcinogenesis	0.745	0.672	0.561	**0.770**	0.707
Financial	0.173	0.126	0.187	**0.317**	0.123
Mondial	**0.776**	0.763	0.723	0.652	0.650
Mutagen.	0.920	**0.971**	0.725	0.902	0.851
Pyrimidine	0.956	**1**	0.819	0.990	0.769
Sisya	**0.708**	0.706	0.706	0.622	0.621
Sisyb	**0.287**	0.286	0.286	0.286	0.286
Triazine	0.560	0.734	0.760	**0.855**	0.685
Yeast	0.428	0.502	0.448	0.469	**0.588**
Bongard	0.899	0.966	**0.970**	0.635	0.300

8 Conclusions

We proposed the PASCAL algorithm for learning probabilistic constraint logic theories from interpretations, a probabilistic extension of integrity constraints (ICs) presented in [1]. PASCAL can exploit either gradient descent or L-BFGS for tuning the parameters. PASCAL is the first system able to learn probabilistic ICs, and has been demonstrated to achieve comparable or better performance in terms of AUC-PR and AUC-ROC than several state-of-the-art sta-

Table 6 Average AUC-ROC. LIFT-EM and LIFT-LBFGS columns show the results for LIFTCOVER using respectively EM and L-BFGS for parameter learning. In bold the best results for each dataset.

Dataset	SLIPCOVER	LIFT-EM	LIFT-LBFGS	PASCAL	TILDE
Bupa	**1**	**1**	**1**	**1**	0.500
Carcinogenesis	0.695	**0.766**	0.472	0.763	0.667
Financial	0.568	0.432	0.535	**0.745**	0.478
Mondial	0.630	**0.663**	0.643	0.495	0.500
Mutagen.	0.826	**0.931**	0.649	0.806	0.778
Pyrimidine	0.925	**1**	0.850	0.993	0.815
Sisya	0.719	0.372	**0.721**	0.502	0.499
Sisyb	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**
Triazine	0.544	0.713	0.760	**0.803**	0.600
Yeast	0.733	0.786	0.721	**0.794**	0.718
Bongard	0.944	0.975	**0.987**	0.749	0.500

Table 7 Average learning time in seconds. LIFT-EM and LIFT-LBFGS columns show the results for LIFTCOVER using respectively EM and L-BFGS for parameter learning. In bold the best results, The running time of TILDE is scaled.

Dataset	SLIPCOVER	LIFT-EM	LIFT-LBFGS	PASCAL	TILDE
Bupa	1.349	**0.243**	1.239	12.324	2.831
Carcinogenesis	25568	**7.850**	76.490	156.711	42.245
Financial	**0.178**	0.235	0.246	12.745	26.630
Mondial	6.490	5.911	3.984	20.139	**2.191**
Mutagen.	12.110	12.770	122.800	19.303	**5.803**
Pyrimidine	54.620	54.990	126.100	48.040	**2.631**
Sisya	45.750	**0.932**	2.252	329.781	1070.457
Sisyb	37.000	**0.226**	0.412	6.339	414.773
Triazine	728.200	56.690	109.100	22.618	**5.236**
Yeast	202.400	**0.502**	69.300	57.856	175.486
Bongard	3.113	20.602	4.278	**2.954**	5.913

Table 8 Average AUC-PR for the systems LEMUR, MLN-BC, MLN-BC with sampling (MLN-BC samp.), MLN-BT and MLN-BT with sampling (MLN-BT samp.), compared to PASCAL. In bold the best results.

Dataset	LEMUR	MLN-BC	MLN-BC samp.	MLN-BT	MLN-BT samp.	PASCAL
Carcinogenesis	0.691	0.619	0.633	0.503	0.494	**0.770**
Mondial	**0.864**	0.585	0.742	0.735	0.781	0.652
Mutagenesis	**0.952**	0.690	0.831	0.872	–	0.902

tistical relational learners, based on probabilistic logic programs or on Markov Logic Networks, in a comparable execution time.

The main limitation of PASCAL is that it performs discriminative learning, not generative learning. As such it is not suitable for building domain models but rather for building predictive models, allowing the classification of interpretations but not capturing the dependencies among ground atoms.

Table 9 Average AUC-ROC for the systems LEMUR, MLN-BC, MLN-BC with sampling (MLN-BC samp.), MLN-BT and MLN-BT with sampling (MLN-BT samp.), compared to PASCAL. In bold the best results.

Dataset	LEMUR	MLN-BC	MLN-BC samp.	MLN-BT	MLN-BT samp.	PASCAL
Carcinogenesis	0.721	0.632	0.641	0.361	0.441	**0.763**
Mondial	**0.782**	0.390	0.594	0.601	0.662	0.495
Mutagenesis	**0.952**	0.553	0.741	0.867	0.823	0.806

Table 10 Average time in seconds for the systems LEMUR, MLN-BC, MLN-BC with sampling (MLN-BC samp.), MLN-BT and MLN-BT with sampling (MLN-BT samp.). The running time of PASCAL is scaled.

Dataset	LEMUR	MLN-BC	MLN-BC samp.	MLN-BT	MLN-BT samp.	PASCAL
Carcinogenesis	11230	55	45	175	181	154.49
Mondial	23435	45	709	114	359	**19.85**
Mutagenesis	22	65	64	1438	2368	**19.03**

Table 11 p-values of a paired two-tailed t-test when comparing the AUC-ROC and AUC-PR of PASCAL with respect to TILDE. In bold when the significance level is smaller than 0.05.

Dataset	TILDE/AUC-ROC	TILDE/AUC-PR
Bupa	**0**	**1.87E-3**
Financial	**4.94E-4**	**8.71E-3**
Mondial	0.783	0.894
Mutagenesis	0.604	0.144
Pyrimidine	0.069	0.058
Sisya	**0.010**	0.191
Triazine	0.106	0.069
Yeast	**2.66E-5**	**1.64E-6**

In the future we plan to apply PASCAL to domains from the field of Business Process Management, which we believe can benefit from a probabilistic evaluation of business workflows.

Another future work, anticipated in the Introduction, regards investigating how to introduce probabilistic reasoning in the framework of interaction protocols in societies of agents, suitably extending the declarative and operational semantics of [2], in order to be able to monitor and verify the compliance of a partial, and still not complete, interpretation (i.e., a narrative of occurred events, but not yet completed).

PASCAL will be also included in the `cplint` framework and made available in the cplint-on-SWISH web application [3] at `http://cplint.eu` to facilitate experimenting with it.

Acknowledgments

We thank the anonymous reviewers for their insightful comments on previous drafts of this article.

This work was partially supported by the "GNCS-INdAM".

References

1. Alberti, M., Bellodi, E., Cota, G., Lamma, E., Riguzzi, F., Zese, R.: Probabilistic constraint logic theories. In: A. Hommersom, S. Abdallah (eds.) Proceedings of the 3nd International Workshop on Probabilistic Logic Programming (PLP), *CEUR Workshop Proceedings*, vol. 1661, pp. 15–28. Sun SITE Central Europe, Aachen, Germany (2016). URL `http://ceur-ws.org/Vol-1661/#paper-02`
2. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the *S*CIFF framework. ACM T. Comput. Log. **9**(4), 29:1–29:43 (2008)
3. Alberti, M., Cota, G., Riguzzi, F., Zese, R.: Probabilistic logical inference on the web. In: G. Adorni, S. Cagnoni, M. Gori, M. Maratea (eds.) AI*IA 2016, *Lecture Notes in Computer Science*, vol. 10037, pp. 351–363. Springer International Publishing (2016). DOI 10.1007/978-3-319-49130-1_26
4. Apt, K.R., Bezem, M.: Acyclic programs. New Generat. Comput. **9**(3-4), 335–363 (1991)
5. Bellodi, E., Riguzzi, F.: Learning the structure of probabilistic logic programs. In: S. Muggleton, A. Tamaddoni-Nezhad, F. Lisi (eds.) 22nd International Conference on Inductive Logic Programming, *LNCS*, vol. 7207, pp. 61–75. Springer Berlin Heidelberg (2012)
6. Bellodi, E., Riguzzi, F.: Structure learning of probabilistic logic programs by searching the clause space. Theor. Pract. Log. Prog. **15**(2), 169–212 (2015). DOI 10.1017/S1471068413000689
7. Blockeel, H., De Raedt, L., Jacobs, N., Demoen, B.: Scaling up inductive logic programming by learning from interpretations. Data Min. Knowl. Discov. **3**(1), 59–93 (1999)
8. Blockeel, H., Raedt, L.D.: Top-down induction of first-order logical decision trees. Artif. Intell. **101**(1-2), 285–297 (1998). URL `https://doi.org/10.1016/S0004-3702(98)00034-4`
9. Bongard, M.M.: Pattern Recognition. Hayden Book Co., Spartan Books (1970)
10. Bruynooghe, M., Mantadelis, T., Kimmig, A., Gutmann, B., Vennekens, J., Janssens, G., De Raedt, L.: Problog technology for inference in a probabilistic first order logic. In: ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings, *Frontiers in Artificial Intelligence and Applications*, vol. 215, pp. 719–724. IOS Press (2010)
11. Cauchy, A.: Méthode générale pour la résolution des systemes d'équations simultanées. Comp. Rend. Sci. Paris **225**(83), 536–538 (1847)
12. Clark, K.L.: Negation as failure. In: Logic and data bases, pp. 293–322. Springer (1978)
13. Davis, J., Goadrich, M.: The relationship between precision-recall and ROC curves. In: ECML 2006, pp. 233–240. ACM (2006)
14. De Raedt, L., Dehaspe, L.: Clausal discovery. Machine Learning **26**(2-3), 99–146 (1997)
15. De Raedt, L., Džeroski, S.: First-Order jk-Clausal Theories are PAC-Learnable. Artif. Intell. **70**(1-2), 375–392 (1994)
16. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: M.M. Veloso (ed.) IJCAI 2007, vol. 7, pp. 2462–2467. AAAI Press/IJCAI (2007)
17. De Raedt, L., Van Laer, W.: Inductive constraint logic. In: ALT 1995, *Lecture Notes in Artificial Intelligence*, vol. 997, pp. 80–94. Springer (1995)
18. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Stat. Soc. B Met. **39**(1), 1–38 (1977)

19. Di Mauro, N., Bellodi, E., Riguzzi, F.: Bandit-based Monte-Carlo structure learning of probabilistic logic programs. Mach. Learn. **100**(1), 127–156 (2015). DOI 10.1007/s10994-015-5510-3
20. Domingos, P., Webb, W.A.: A tractable first-order probabilistic logic. In: J. Hoffmann, B. Selman (eds.) Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12). AAAI Press (2012)
21. Flach, P.A., Lachiche, N.: Naive Bayesian classification of structured data. Machine Learning **57**(3), 233–269 (2004). DOI 10.1023/B:MACH.0000039778.69032.ab. URL http://dx.doi.org/10.1023/B:MACH.0000039778.69032.ab
22. Gavanelli, M., Lamma, E., Riguzzi, F., Bellodi, E., Zese, R., Cota, G.: An abductive framework for datalog± ontologies. In: M.D. Vos, T. Eiter, Y. Lierler, F. Toni (eds.) Technical Communications of the 31st International Conference on Logic Programming (ICLP 2015), *CEUR Workshop Proceedings*, vol. 1433. CEUR-WS.org (2015)
23. Gordon, D.M.: A survey of fast exponentiation methods. J. Algorithms **27**(1), 129 – 146 (1998). DOI 10.1006/jagm.1997.0913
24. Kapur, D., Narendran, P.: Np-completeness of the set unification and matching problems. In: International Conference on Automated Deduction, pp. 489–495. Springer (1986)
25. Khot, T., Natarajan, S., Kersting, K., Shavlik, J.W.: Learning Markov Logic Networks via functional gradient boosting. In: Proceedings of the 11th IEEE International Conference on Data Mining, pp. 320–329. IEEE (2011)
26. Kok, S., Domingos, P.: Learning Markov Logic Networks using structural motifs. In: J. Fürnkranz, T. Joachims (eds.) ICML 2010, pp. 551–558. Omnipress (2010)
27. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: 18th International Conference on Machine Learning, vol. 1, pp. 282–289 (2001)
28. Lamma, E., Mello, P., Riguzzi, F., Storari, S.: Applying inductive logic programming to process mining. In: Proceedings of the 17th International Conference on Inductive Logic Programming, ILP 2007, no. 4894 in Lecture Notes in Artificial Intelligence, pp. 132–146. Springer, Heidelberg, Germany (2008). DOI 10.1007/978-3-540-78469-2_16. URL http://dx.doi.org/10.1007/978-3-540-78469-2_16
29. Mihalkova, L., Mooney, R.J.: Bottom-up learning of Markov logic network structure. In: Proceedings of the 24th International Conference on Machine Learning, pp. 625–632. ACM (2007)
30. Muggleton, S.: Inverse entailment and Progol. New Generat. Comput. **13**, 245–286 (1995)
31. Neville, J., Jensen, D.D., Friedland, L., Hay, M.: Learning relational probability trees. In: L. Getoor, T.E. Senator, P.M. Domingos, C. Faloutsos (eds.) Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003, pp. 625–630. ACM Press (2003)
32. Nguembang Fadja, A., Riguzzi, F.: Lifted discriminative learning of probabilistic logic programs. Machine Learning (2018). DOI 10.1007/s10994-018-5750-0. URL https://doi.org/10.1007/s10994-018-5750-0
33. Nguembang Fadja, A., Riguzzi, F.: Lifted discriminative learning of probabilistic logic programs. Machine Learning **108**(7), 1111–1135 (2019). DOI 10.1007/s10994-018-5750-0. URL http://ml.unife.it/wp-content/uploads/Papers/NguRig-ML18.pdf
34. Nocedal, J.: Updating Quasi-Newton matrices with limited storage. Math. Comput. **35**(151), 773–782 (1980)
35. Poole, D.: Logic programming, abduction and probability - a top-down anytime algorithm for estimating prior and posterior probabilities. New Generat. Comput. **11**(3), 377–400 (1993)
36. Poole, D.: The Independent Choice Logic for modelling multiple agents under uncertainty. Artif. Intell. **94**, 7–56 (1997)
37. Poole, D., Zhang, N.L.: Exploiting contextual independence in probabilistic inference. J. Artif. Intell. Res. **18**, 263–313 (2003)
38. Provost, F.J., Fawcett, T.: Robust classification for imprecise environments. Mach. Learn. **42**(3), 203–231 (2001)

39. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. **62**(1-2), 107–136 (2006)
40. Riguzzi, F., Swift, T.: Tabling and answer subsumption for reasoning on logic programs with annotated disjunctions. In: ICLP TC 2010, *LIPIcs*, vol. 7, pp. 162–171. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010). DOI 10.4230/LIPIcs.ICLP.2010.162
41. Riguzzi, F., Swift, T.: Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics. Theor. Pract. Log. Prog. **13**(2), 279–302 (2013). DOI 10.1017/S1471068411000664
42. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: L. Sterling (ed.) ICLP 1995, pp. 715–729. MIT Press (1995)
43. Sato, T., Kameya, Y.: PRISM: a language for symbolic-statistical modeling. In: IJCAI 1997, vol. 97, pp. 1330–1339 (1997)
44. Singla, P., Domingos, P.: Discriminative training of Markov logic networks. In: 20th National Conference on Artificial Intelligence (AAAI 2005), pp. 868–873. AAAI Press/The MIT Press (2005)
45. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: B. Demoen, V. Lifschitz (eds.) ICLP 2004, *Lecture Notes in Computer Science*, vol. 3131, pp. 431–445. Springer (2004). DOI 10.1007/978-3-540-27775-0_30
46. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theor. Pract. Log. Prog. **12**(1-2), 67–96 (2012). DOI 10.1017/S1471068411000494