Probabilistic Rule Learning in Nonmonotonic Domains

Domenico Corapi[1], Daniel Sykes[1], Katsumi Inoue[2], and Alessandra Russo[1]

[1] Department of Computing
Imperial College London
180 Queen's Gate, SW7 2AZ
London, UK
{d.corapi,d.sykes,a.russo}@imperial.ac.uk
[2] National Institute of Informatics
Chiyoda-ku, 2-1-2, Hitotsubashi
Tokyo 101-8430, Japan
ki@nii.ac.jp

Abstract. We propose here a novel approach to rule learning in probabilistic nonmonotonic domains in the context of answer set programming. We used the approach to update the knowledge base of an agent based on observations. To handle the probabilistic nature of our observation data, we employ parameter estimation to find the probabilities associated with each of these atoms and consequently with rules. The outcome is the set of rules which have the greatest probability of entailing the observations. This ultimately improves tolerance of noisy data compared to traditional inductive logic programming techniques. We illustrate the benefits of the approach by applying it to a planning problem in which the involved agent requires both nonmonotonicity and tolerance of noisy input.

Keywords: Inductive Logic Programming, Probabilistic Logic Programming, Answer Set Programming, Hypothetical Reasoning, Planning.

1 Introduction

Traditional machine learning techniques assume preliminary knowledge about the domain elements which are critical to solve the learning task. This is often expressed as a task of finding a target function from a collection of examples, where the examples are assumed to be useful and relevant to the learned function. This is problematic in certain contexts. For example, a robot whose goal is to deliver an object to a location may fail to achieve this goal for a variety of reasons. Perhaps the motors failed, or the object was dropped along the way. It is not immediately obvious how to relate the observation (a failure) to the knowledge base of the robot. In other words, the extraction of features relevant to the learning task is itself a problem that must be considered.

Inductive logic programming (ILP) [16] is a technique, using logic programs as a representation language, that can be seen as a general-purpose method to find the relevant features amongst a large set of candidates [6] through a process of generalisation. The dependencies between features in the knowledge base and observations are then captured in a logic program. Such logic programs, when explicitly accounting for nonmonotonicity, are expressive enough to be used for a wide range of applications, e.g. for planning tasks [29].

However, ILP techniques tend to produce poor results in domains where a logical representation alone cannot capture uncertainty since they are heavily reliant upon the concept of logical implication, which limits their capacity to cope with erroneous training examples and knowledge.

We can improve upon this dependency by recognising that the example data and the knowledge available about the domain naturally suffer from noise. Explicit probabilistic treatment of the observations and the knowledge also has the advantage of producing 'higher' abstraction as insignificant aspects of the examples are ignored.

In this work, we develop a novel rule learning approach, which builds on an existing technique that transforms ILP to abductive reasoning [4] (enabling us to handle nonmonotonic domains) and techniques for estimating probabilities of logic facts [10], [11]. Answer set programming (ASP) [17] is used to generate candidate rules for the ILP problem, which is to say, rules that logically entail some of the observations. Gradient descent is then used to estimate probabilities for the rules so that the error between the expected probability of the observations (as calculated) and the actual probability is minimised. The set of rules that most closely fits the observations is given as the result. Our main contribution is to show how through an encoding technique that represents logic rules as logic atoms, established techniques can be adapted to derive structured rules that improve a nonmonotonic theory, together with a probabilistic weight by estimating the probabilities of these logic atoms.

We have applied our approach in the context of learning rules to perform planning for a reactive agent. While reactive planning already has the benefit of making choices on the basis of sensing during execution, there remain situations when multiple actions can be performed in the same state. Standard reactive planning chooses one such action arbitrarily, but in the uncertain physical environments in which planning is often used, greater reliability can be achieved by choosing the actions which are most likely to lead to success of the overall plan.

The rest of the paper is organised as follows. Section 2 introduces some preliminary definitions. Section 3 describes the approach (NoMPRoL) in detail. Section 4 introduces our planning case study and the results gained thereof. Section 5 discusses the contribution and related work. Section 6 describes future work and concludes.

2 Preliminaries

We assume that the reader is familiar with logic programming [18]. Given a logic-based alphabet consisting of variables, constants and predicates, an *atom* is an expression of the form $p(t_1, .., t_n)$, where p is a predicate and t_i are terms (variable or ground) in the alphabet. An atom can be negated using *negation as failure*. *Literals* are atoms a or negated atoms $\text{not } a$. We say that $\text{not } a$ is true if we cannot find evidence supporting the truth of a. Atoms and literals are used to create *rules* of the form: $a \leftarrow b_1, ..., b_m, \text{not } c_1, ..., \text{not } c_n$, where a, b_i and c_j are atoms. Intuitively, this means *if all atoms b_i are known/true and no atom c_i is known/true, then a must be known/true.*

We refer to a as the head and $b_1, ..., b_m, \text{not } c_1, ..., \text{not } c_n$ as the body of the rule. A *(normal) logic program (or theory)* is a conjunction of rules and is also denoted by a set of rules. The semantics is defined in terms of *answer sets* [17], i.e. assignments of true and false to all atoms in the program that satisfy the rules in a minimal and consistent fashion. A program has zero or more answer sets, each corresponding to a solution.

Our approach makes use of abductive reasoning, i.e. reasoning about explanations for given observations. An *abductive logic program* is a pair $\langle \mathcal{T}, A \rangle$ where $\mathcal{T}$ is a theory, and A is set *abducible* atoms. An abductive solution Δ is the subset of abducibles that must be added to the theory in order to obtain a particular answer set. We refer to the answer sets M associated with some answer Δ as *generalised answer sets* [12].

Definition 1. *Let $\langle \mathcal{T}, A \rangle$ be an abductive logic program and $\Delta \subseteq A$ be an abductive solution. M is a* generalised answer set *for $\langle \mathcal{T}, A \rangle$ iff M is an answer set of $\mathcal{T} \cup \Delta$.*

We use the notation Δ_M to denote the abductive solution associated with generalised answer set M, i.e. $\Delta_M = A \cap M$. $AS(\mathcal{T}, A)$ denotes all the generalised answer sets of $\langle \mathcal{T}, A \rangle$. Given an abductive logic program $\langle \mathcal{T}, A \rangle$, we assume that every possible $\mathcal{T} \cup \Delta$ has at most one answer set[1].

3 Approach

We present a general methodology that, given an existing knowledge base in the form of a logic program $\mathcal{T}$, and a set of observations X subject to noise, finds the set of rules with estimated probabilities that explain the observations. The approach, which we call NoMPRoL, is divided into three stages as shown in Figure 1. In the first stage, the task of finding rules for the given knowledge base is encoded as an abductive reasoning problem. A transformed knowledge base (with the observations) is then presented to an answer set solver which finds all solutions (i.e. models) of this input, which is to say, in addition to atoms relating to the original knowledge base, answer sets contain abducibles representing learned rules that entail the observations. By construction, as clarified in Section 3.1, such abductive solutions can be transformed into a set of rules whilst preserving the semantics. The third part of the approach is to estimate probabilities for abducibles to find a maximum likelihood hypothesis comprising the set of rules H and a probability distribution P_0^θ. This is achieved by using gradient descent to minimise a mean squared error function.

We define a probability distribution over a set of abductive solutions, instantiating the framework of [27], treating negation in a similar way as in [23].

Definition 2. *A* probabilistic theory *is a tuple $\langle \mathcal{T}, A, P_0^\theta \rangle$ where $\mathcal{T}$ is a theory, A is a set of abducible atoms and P_0^θ is a probability distribution over 2^A.*

P_0^θ depends on $|A|$ independent variables θ_a associated with the probability that the abducible atom $a \in A$ is true. We call the set of all such variables θ. Note that the independence assumption is common in the existing frameworks for probabilistic logic.

[1] This is true whenever $\mathcal{T}$ is acyclic. Under this assumption the unique answer set is characterised by the Clark's completion [3] of the program.

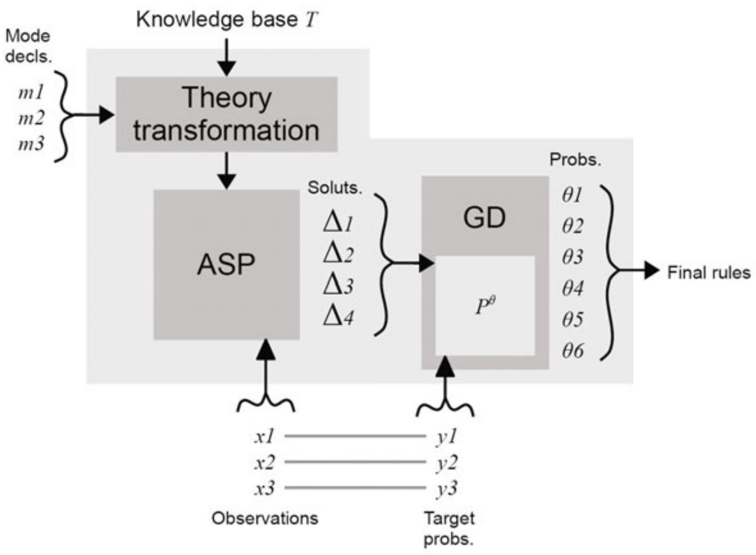

Fig. 1. Schematic view of the three phases in NoMPRoL

Mutual exclusivity of the variables can be modelled through integrity constraints (that in this work extend the *independent choices* in [23]). For any $\varDelta \subseteq A$,

$$P_0^\theta(\varDelta) = \prod_{a \in \varDelta} \theta_a \prod_{a \in A \setminus \varDelta} (1 - \theta_a)$$

defines the probability of sets of abducibles and indirectly of models that contain them. We define the probability of a logic literal as follows:

$$P^\theta(l|\langle \mathcal{T}, A\rangle) = \frac{\sum_{\{M: M \in AS(\mathcal{T},A), l \in M\}} P_0^\theta(\varDelta_M)}{\sum_{\{M: M \in AS(\mathcal{T},A)\}} P_0^\theta(\varDelta_M)} \tag{1}$$

The probability of a literal l is given by the sum of the probabilities of the abductive solutions that entail l normalised over the sum of the probabilities of all abductive solutions of the theory. The probabilities always implicitly refer to an underlying abductive logic program. If the abductive theory is clear from the context we use the notation $P^\theta(l)$.

3.1 ILP as Abductive Reasoning

ILP is used when a certain knowledge base must be enriched with rules that are also able to classify new examples. We follow the common practice of defining the space of acceptable rules (the *language bias*) with a set of mode declarations. *Head* and *body mode declarations* L [20] define the structure of the atoms (by means of a $schema$) that can appear in the body or in the head of the rules, hence defining the space $s(L)$ of possible hypotheses (the language bias).

Mode declarations are specified as $m : mode(t, r(m), s)$ where m is the label; the first argument specifies whether it is a head (h) or body (b) mode declaration; the second argument specifies the maximum number of occurrences of m; and the third argument specifies the *schema*. A schema s is a ground literal that contains one or more placemarkers. A *placemarker* is a ground function with one of the three symbols '+' for *input placemarkers*, '-' for *output placemarkers*, '#' for *constant placemarkers*. The only argument of the function is a constant called *type*.

Let r be a clause h :- $b_1, ..., b_n$ and L be a set of mode declarations. Then r is *compatible* with L (i.e. $r \in s(L)$) iff the following conditions are met:

1. h corresponds to the schema of a head mode declaration in L where all the input placemarkers and output placemarkers are replaced with variables and all the constant placemarkers are replaced with constants;
2. for each $b_i, i = 1, ..., n$, b_i corresponds to the schema of a body mode declaration in L where all the input placemarkers and output placemarkers are replaced with variables and all the constant placemarkers are replaced with constants;
3. every variable that replaces a input placemarker in any of the literals in the body either replaces a input placemarker in h or an output placemarker in some atom $b_j, j < i$.

We provide an example and refer to [20] for details.

Example 1. Consider the following mode declarations L:

$$\begin{aligned}&m1 : mode(h, 1, daughter(+person, +person)).\\&m2 : mode(b, 1, mother(+person, +person)).\\&m3 : mode(b, 1, sex(+person, \#mf)).\end{aligned}$$

Given a background theory $\mathcal{T}$

$$\begin{aligned}&mf(m).\ \ mf(f).\\&sex(ann, f).\ \ sex(tom, m).\\&mother(mary, ann).\end{aligned}$$

and a set of constraints $\mathcal{R} = \{:\text{-}\ not\ daughter(ann, mary), :\text{-}\ daughter(tom, mary)\}$, the following labelled rules are compatible with L (i.e. $\{r1, r2\} \subseteq s(L)$):

$$\begin{aligned}&r1 : daughter(X, Y) :\text{-}\ sex(X, m).\\&r2 : daughter(X, Y) :\text{-}\ mother(X, Y), sex(Y, m).\end{aligned}$$

and the following rule:

$$r3 : daughter(X, Y) :\text{-}\ mother(Y, X), sex(X, f).$$

is such that $\mathcal{T} \cup \mathcal{R} \cup \{r3\}$ has one generalised answer set (while $\mathcal{T} \cup \mathcal{R}$ has none) and $r3 \in s(L)$. Thus $r3$ is a possible solution and it defines the concept of $daughter$ based on the predicates $mother$ and sex, consistently with the constraints. Rules $r1$ and $r2$ are within the language bias but violate the given constraints (in turn they imply $daughter(tom, mary)$ and $not\ daughter(ann, mary)$).

We treat examples as constraints, so the task at hand is to find new rules that satisfy a set of given constraints. This is achieved by "lifting" the *abductive* reasoning to *inductive* reasoning using a similar representation to that in [4]. In contrast to [4], we use ASP mainly because it better supports the generation of all the solutions and the use of integrity constraints. To transform an ILP task into an abductive search task we represent all the rules within $s(L)$ as logical atoms that can be abduced using an answer set solver. Abductive solutions for the new theory can be transformed back into a solution for the equivalent ILP problem. The correctness and completeness of the transformations are obtained, as in [4], using logical equivalences after the truth values of the abducibles are defined.

For a given mode declaration, $s*$ denotes the atom created from the schema s by replacing the placemarkers with newly defined variables. $ins(s*)$ is shorthand for a list of all the variables that replace input placemarkers, $outs(s*)$ is a list of the variables that replace output placemarkers and $cons(s*)$ is a list of the variables that replace constant placemarkers in $s*$. All abducibles are marked by a \$ character.

Intuitively we want to transform the original theory into a "meta-theory" that can be used to reason about possible inductive solutions. The transformation is such that whenever certain abducibles are part of an abductive solution we must infer that a certain head is needed in a rule in inductive solution. Other abducibles are derived to define the body of such rules. Atoms of the type $\$head(h, id)$ are abduced whenever a rule (identified by id) is needed that uses the mode declaration h in the head. Atoms of the type $\$body(b, id, r, l, c)$ are abduced whenever a condition that uses the mode declaration b is required in the body of a rule that is part of a solution. In $\$body(b, id, r, l, c)$, r disambiguates amongst different uses of the same mode declaration in the body; l defines the bindings of the input variables; and c contains the constants used. For brevity we will not go into details of the type checking. Given a set of mode declarations L, we construct a set of rules $\mathcal{R}$ as follows:

– Let $b_1, ..., b_n$ be labels for all body mode declarations in L. For each head mode declaration $m : mode(h, r(m), s)$ the following clause is in $\mathcal{R}$:

$$\begin{array}{ll} s* \text{ :- } & \$head(m, ID), \\ & not\ body(b_1, ID, 1, ins(s*), X_{1,1}), \\ & \ldots \\ & not\ body(b_1, ID, r(b_1), X_{1,(r(b_1)-1)}, X_{1,r(b_1)}), \\ & not\ body(b_2, ID, 1, X_{1,(r(b_1))}, X_{2,1}), \\ & \ldots \\ & not\ body(b_2, ID, r(b_2), X_{2,(r(b_2)-1)}, X_{2,r(b_2)}), \\ & \ldots \\ & not\ body(b(n), ID, r(b_n), X_{n,(r(b_n)-1)}, X_{n,r(b_n)}). \end{array}$$

These rules represent a skeleton for the rules that can be considered as inductive hypotheses. If the $\$head$ atom is not part of an abductive solution the rule is always true and has no effect on the rest of the theory. Otherwise the truth of the head depends on the other conditions. In each *body* condition, except the first, the fourth argument is the same variable as the fifth argument of the previous *body* condition (ordered left to right). This makes it possible to share variables between conditions and bind input variables to output variables. Variables that substitute output place-markers are this way shared amongst conditions. Note that an order is established *a priori* mainly for efficiency purposes and to remove redundancies. As shown in [4] this can be avoided with a different encoding.

– For each body mode declaration $m : mode(b, r(m), s)$, the following clause is in $\mathcal{R}$:

$$\begin{aligned}&body(m, ID, R, V, X) \text{ :- }\\&\quad link(V, ins(s*), E),\\&\quad append(V, outs(s*), X),\\&\quad \$body(m, ID, R, E, cons(s*)),\\&\quad not\ s* .\end{aligned}$$

Rules of this type only have effect on the transformed theory if the $\$body$ atom is abduced. The atom $link(v, i, e)$ produces a list i where all the elements are also in v and a list of e that contains the indexes of the elements i in v (e.g. $link((ann, mary, f, tom), (ann, f), (1, 3))$ is a true instance). The list in the third argument is used in the abducible $\$body$ to codify the binding of the variables. *append* is a conventionally defined operator for appending lists.

The theory $\mathcal{R}$ is used within the learning process and processed together with a background theory and a set of constraints by the ASP solver. The final solution is transformed back into a set of rules.

Example 2. Let $\mathcal{T}$, $\mathcal{R}$ and L be the theories and mode declarations from the previous example. $\mathcal{R}$ is constructed from L as follows:

$$\begin{aligned}&daughter(X, Y) \text{ :- } \$head(m1, ID),\\&\quad not\ body(m2, ID, 1, (X, Y), X_{m2,1}),\\&\quad not\ body(m3, ID, 1, X_{m2,1}, X_{m3,1}).\end{aligned}$$

$$\begin{aligned}&body(m2, ID, 1, V, X) \text{ :- }\\&\quad link(V, (Xl, Yl), L),\\&\quad \$body(m2, ID, 1, L, ()),\\&\quad not\ mother(Xl, Yl).\end{aligned}$$

$$\begin{aligned}&body(m3, ID, 1, V, X) \text{ :- }\\&\quad link(V, (Xl), L),\\&\quad \$body(m3, ID, 1, L, C),\\&\quad not\ sex(Xl, C).\end{aligned}$$

Using an ASP solver we can generate the following abductive solution that codifies rule $r3$ from the previous example $\Delta = \{\$head(m1, r3), \$body(m2, r3, 1, (2, 1), ()), \$body(m3, r3, 1, (1), (f))\}$. The first abducible specifies mode declaration $m1$ is used in the head of $r3$. The second specifies $m3$ is used in the body; the list $(2, 1)$ specifies that the first input variable (left to right in order of appearance in the mode declaration) is linked to the second variable in the head and that the second input variable is linked to the first variable in the head. The third abducible specifies $m3$ is used in the body, that the input variable is linked to the first variable in the head and that the constant is instantiated to f.

As made clear in Section 4.3, by defining a probability distribution over the abducibles we can instantiate a probability for the literals in the constraints. For example, let $\theta_{\$head(m1,r3)} = 1$, $\theta_{\$body(m2,r3,1,(2,1),())} = 0.8$, $\theta_{\$body(m3,r3,1,(1),(f))} = 1$ and $\theta_a = 0$ for all $a \in A \setminus \Delta$. In this case $P_0{}^{\theta}(\Delta) = 0.8$ and $P_0{}^{\theta}(\Delta \setminus \{\$body(m2,r3,1,(2,1),())\}) = 0.2$. The two considered sets of abducibles together with $\mathcal{R}$ are respectively logically equivalent to $\{daughter(X,Y)$:- $mother(Y,X), sex(X,f)\}$ and $\{daughter(X,Y)$:- $sex(X,f)\}$ and implicitly define their probabilities.

3.2 Model Generation

Given a set of mode declarations L, the theory $\mathcal{R}$ generated from it, and a background theory $\mathcal{T}$, for each observation x we generate all the generalised answer sets that are needed to calculate $P^{\theta}(x)$. That is to say, we generate all generalised answer sets for the abductive theory $\langle \mathcal{T} \cup \{: -not\ x\} \cup \mathcal{R}, A\rangle$. A includes all the atoms with predicate $\$head$ and $\$body$ that correspond to meaningful literals in the final rule. The solutions $\Delta_1, ..., \Delta_q$ are used to construct the data structures that codify the sum of products in Equation (1), where each Δ_i corresponds to one element of the sum. Conceptually, the abductive solutions correspond to a boolean formula where all the abducibles within a solution are in conjunction (and negated if not part of the model) and all the solutions are in disjunction.

3.3 Parameter Estimation

The observations x_i are represented by literals and they are associated with a target probability y_i, i.e. the learned theory should entail each observation x_i with probability y_i. We assume that the true target value is altered by random Gaussian noise to account for errors in the background theory and noise in the observed literals as in [10]. Finding the maximum likelihood (abductive) hypothesis under this assumption corresponds to minimising the following mean squared error function [19]:

$$MSE(\theta) = \frac{1}{|X|}\sum_i (y_i - P^{\theta}(x_i|\langle \mathcal{T}, A\rangle))^2 \qquad (2)$$

We use gradient descent[2] over equation (2) to estimate the probabilities of each abducible. In our particular case, positive and negative examples are modelled as (possibly negated) literals with target probability $y_i = 1$. Initially θ is given random values. Our implementation uses the algorithm described in [10], adopting binary decision diagrams (BDDs) to calculate the value of the gradient and of the probabilities of the observations at each iteration. The final θ_o that minimises the MSE can be mapped into a set of probabilistic hypotheses, thus providing rules that improve the original knowledge base by taking account of the probabilistic observations. This can be seen in more detail in Section 4.3.

[2] Other optimisation algorithms can be used.

4 Case Study: A Planning Agent

An ideal application for nonmonotonic rule learning is a planning problem [9] in which the initial knowledge base (from which plans are generated) is discovered to be incorrect or incomplete when the plans are generated and executed. Moreover, when the plans refer to a robotic agent situated in the physical world, with all its uncertainties, a probabilistic approach is required. Probabilities can also account for an incomplete model of the environment or of other agents acting in the same environment.

4.1 Knowledge Base

Figure 2 depicts the feedback loop between the knowledge base and the running system. The knowledge base (also known as a *domain model*) contains a description of the agent and the world it occupies, expressed as logic program in terms of conditions that hold and actions that can be performed (similar to Event Calculus [15]). Figure 3 shows part of the knowledge base for our specific example of a mobile robot, which states that a move action between two locations is possible if the places are connected and the robot is at the initial location.

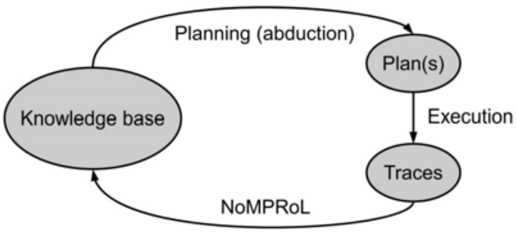

Fig. 2. Feedback between planning and rule learning

Typically there are many sequences of actions which the agent could perform to achieve a given goal. In a traditional linear plan, a single sequence is selected and the agent performs it blindly. *Reactive* or *universal* plans [28] improve robustness by having the agent sense the new world state after each action so that unexpected state changes—those not described by the knowledge base—can be handled. However, even for a given state there are often several actions which lead to the goal, and reactive planners make an almost arbitrary choice (often the shortest path). The practical truth is that each action has a different probability of leading to the goal, and it is on this basis that alternatives should be selected. The problem described is nonmonotonic as adding rules to the theory can invalidate previous consequences. The case study provided is a small instance of a class of problems that can involve complex interactions with the environment and rich contextual properties to be taken into account, e.g. position of other agents, speed, available power.

We generate plans given a goal condition such as $holdsAt(at(gps5), 3)$ by following an abductive procedure [29]. The resulting linear plans are merged to produce (a variant

```
:-  do(E1, T),
    do(E2, T),
    E1 != E2.

linked(gps1, gps2).
linked(gps1, gps3).
linked(gps1, gps4).
...

%move
possible(move(L1, L2), T) :- L1 != L2,
                             holdsAt(at(L1), T),
                             linked(L1, L2).
initiates(move(L1, L2), at(L2), T).
terminates(move(L1, L2), at(L1), T).
possible(move(L1, L2), T) :- L1 != L2,
                             holdsAt(at(L1), T),
                             linked(L1, L2).
...
happens(E, T) :- do(E, T),
                 succeeds(E,T).
```

Fig. 3. Knowledge base fragment

of) a reactive plan in which each state is associated with one or more actions, thus preserving the choice of paths present in the knowledge base. Each possible action for the current state has a probability (of achieving the goal) given by the sum over the probabilities of possible worlds (represented by generalised answer sets) that contain the action. The action with the highest probability is chosen at each iteration.

4.2 Trace Generation

Traces, from which the observations are derived, are generated by executing the reactive plan as follows:

1. the current environment state is sensed;
2. action(s) applicable to the current state are generated abductively for the given goal; and
3. one action is executed. Initially, when no probabilistic bias is available, where there is a choice of actions (multiple paths to the goal), one is selected randomly.

These steps are repeated until the goal condition is met, or a time-out occurs. At each iteration, the sensed state and the action chosen are recorded. The sequence of states and actions provides a trace. Figure 4 shows an example trace.

From each trace j we derive a set of observations $\{o_{1,j}, ..., o_{m,j}\}$ (one for each time the environment is sensed), and a set of *histories* $\{O_{1,j}, ..., O_{m,j}\}$. Each history $O_{i,j}$ includes all the states and actions in the execution up to the time point before the one

```
%trace 12
holdsAt(at(gps1),0).
do(move(gps1,gps2),0).
holdsAt(at(gps2),1).
do(move(gps2,gps5),1).
holdsAt(at(gps5),2).
do(pickup,2).
holdsAt(at(gps5),3).
```

Fig. 4. Execution trace

$o_{i,j}$ refers to. The training set X thus contains such pairs $(o_{i,j}, O_{i,j})$ and the target probability for each observation is set to 1. For example, the trace in Figure 4 gives a set of observations including:

$$o_{12,2} : \begin{cases} holdsAt(at(gps5), 2), \\ not\ holdsAt(holdingBall, 2). \end{cases}$$

$$O_{12,2} : \begin{cases} holdsAt(at(gps1), 0). \\ do(move(gps1, gps2), 0). \\ holdsAt(at(gps2), 1). \\ do(move(gps2, gps5), 1). \end{cases}$$

We are interested here in a revised theory that fits the observations for a given history. The probability associated with each observation thus involves the corresponding history. Consequently, the function we want to minimise is the following:

$$MSE(\theta) = \frac{1}{|X|} \sum_{i,j} (1 - P^{\theta}(o_{i,j} | \mathcal{T} \cup \Delta \cup O_{i,j}))^2 \tag{3}$$

4.3 Experiment

The practical experiment involves a mobile robot carrying a ball and a robotic arm which can remove the ball from the robot. The goal of the system is for the mobile robot to transport the ball to the arm whereupon the ball can be placed in a store next to the arm. This is expressed as $holdsAt(at(gps5), T)$, $holdsAt(holdingBall, T)$. The mobile robot is able to sense obstacles using short-range infra-red sensors and to detect when the ball is placed on top. In order to navigate through the environment, one or more cameras are placed on the ceiling which use simple computer vision techniques to determine the position and orientation of the mobile robot. The robotic arm is equipped with another camera to locate the ball, and pressure sensors to detect when it has successfully gripped an object.

As with any robotic system, there are many sources of noise. The infra-red sensors can detect obstacles incorrectly, causing erratic motion. Likewise, noise in the camera image can lead to incorrect localisation of the robot. Erratic motion may ultimately lead to complete failure of the plan, if for instance the robot collides with an obstacle or goes

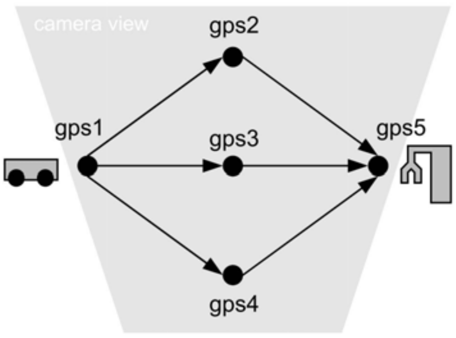

Fig. 5. The environment has three paths from the starting point to the arm

outside the area visible to the cameras. Noise in the camera and pressure sensors of the robotic arm can lead to the ball being dropped or not being identified at all.

To simplify navigation from the starting location to the target (the location of the arm), we create three waypoints, as shown in Figure 5, providing three alternative paths to the arm. The choice of a path leads to different rates of success, as a consequence of the sources of noise described above.

We executed the plan a total of 30 times, each starting in a random state and producing one trace. Each trace consists of the state observed and action taken at each time step, a flag indicating whether or not the goal was achieved and the time taken to execute the plan. Failure is detected explicitly (if the robot leaves the camera range) or otherwise through a time-out. Figure 4 shows one trace in which the robot started at location $gps1$, moved to $gps2$ followed by $gps5$. The absence of $holdsAt(holdingBall, 3)$ shows that the arm failed to pick up the ball, and this trace is recorded as a failure. Finally, we applied NoMPRoL, using the following mode declarations:

$$\begin{array}{l}
mode(h, 2, succeeds(pickup, +time)).\\
mode(h, 2, succeeds(move(+loc, +loc), +time)).\\
mode(b, 2, +loc = \#loc).\\
mode(b, 1, holdsAt(at(+loc, +time))).\\
mode(b, 1, linked(+loc, +loc)).\\
mode(b, 1, wasAt(+loc, +loc)).
\end{array}$$

Results. The benefit of the approach will be shown by whether the robot improved its performance, measured as time taken to reach the goal in the real world and as the ratio of failures over many runs. We run NoMPRoL leaving one trace out for testing each time and using the remaining ones for the training. We then selected only those traces which contain the same actions as those which the agent would perform after learning, thus omitting those where learning has caused the behaviour to change. The average of those execution times and the number of time-outs for the initial and the trained agent are reported in Table 1.

Table 1. Performance measure of the initial and trained agents

	Average execution time (s)	Failure rate
Initial agent	85.2	0.3
Trained agent	63.6	0.1

We consider one representative training instance and discuss the process and results. After the GD algorithm converges each abducible in A is associated with a probability. For example the abducible $body(b\_not\_was\_at, (dp, 1), 1, (1), (gps2))$ has probability $\theta_{body(b\_not\_was\_at,(dp,1),1,(1),(gps2))} = 0.7315$. Intuitively this probability encodes the relevance of the condition within the rule labelled as $(dp, 1)$. Processing and approximating the probabilities for ease of exposition, we obtain the following rules:

$$\begin{array}{lll} r1 : succeeds(pickup, T) \text{ :- } & & \\ & not\ wasAt(gps2, T). & 0.7 \\ & & \\ r2 : succeeds(move(L1, L2), T) \text{ :- } & & \\ & linked(L1, L2). & 0.5 \\ & & \\ r3 : succeeds(move(L1, L2), T) \text{ :- } & & \\ & not\ wasAt(gps3, T), & 0.2 \\ & L2 = gps3, & 0.4 \\ & holdsAt(L1, T), & 1 \\ & linked(L1, L2). & 1 \end{array}$$

The probability associated with a condition intuitively represents the probability that the condition is part of a sampled theory, resulting in a distribution over possible subtheories[3]. In general the final output will be chosen based on the interpreter or based on readability and accuracy requirements.

In our experiment the MSE for all the executions is lower than 1. Non-probabilistic logic rules would never result in a lower MSE, since at least one of the observations would not be entailed. Also, for systems where the estimation is performed on logic facts [11] the dependency between the final pickup and the robot's moving through location $gps2$ would not be detected.

[3] In fact, from these rules we can derive an equivalent distribution over theories, similarly to [5], that are obtained by dropping some of the conditions. For example the following theory has a probability $p = 0.168$ (causing every atom that is entailed by this theory to have a probability greater than p):

$$\begin{array}{l} succeeds(pickup, T) \text{ :- } not\ wasAt(gps2, T). \\ succeeds(move(L1, L2), T) \text{ :- } linked(L1, L2). \\ succeeds(move(L1, L2), T) \text{ :- } holdsAt(L1, T), linked(L1, L2). \end{array}$$

Intuitively, the probability is derived from the product of the probabilities associated with the conditions that are kept and the complement of the probabilities of the conditions that are discarded.

5 Discussion and Related Work

Much of the recent work in the field of Statistical Relational Learning [8] shares similar objectives to those addressed here. The work we present is, to the best of our knowledge, the first with a methodology for learning probabilistic rules in nonmonotonic domains. Under the assumption of monotonicity, the problem of learning probabilistic rules has been addressed separately as an estimation problem and as structure learning [22]. [21] concentrates on single-clause hypotheses; [10] applies gradient descent to estimate probabilities of logic facts in a logic program; [7] extends an established ILP search algorithm with a probability-based search heuristic. [31] addresses the problem of learning probabilistic planning rules.

Markov Logic Networks (MLN) [25] extend first-order logic with Markov networks and are successfully supported by a number of different learning techniques. Amongst these, [30] employs a gradient-driven estimation of the parameters and [14] proposes an integrated solution for structure learning. The main disadvantage of MLN compared to the type of representation used in this paper and, in general, to probabilistic logic representations based on distribution semantics is that formula weights in MLN are counterintuitive and their effect on the inference is not obvious.

Compared to solutions based on Markov Decision Processes (MDP) (e.g. [2]) the approach proposed employs a full first-order representation. An MDP could be used as a target representation for the learned output, with the caveat that maintainability is lost in a more constrained knowledge representation language. Furthermore in the case of planning the states and the effects of the actions can be modelled to take into account, as shown in the example, of relevant past events and states, thus defining the probability of an action at a time point t also as a function of states at $t' \leq t-1$.

The methodology proposed is not a full solution to the problem of learning probabilistic nonmonotonic logic programs. Application in real scenarios would require a considerable computational effort. Nevertheless we provide a general mechanism to transform an inference problem into an equivalent learning task, thus enabling the use of established techniques, and a framework for further developments. ASP tools provide the most effective way to generate and check models for nonmonotonic theories and techniques for parameter estimation benefit from years of consolidation. To improve efficiency, we are currently working on an extension of NoMPRoL that makes use of stochastic sampling techniques as in [13].

We designed an integrated mechanism to solve the problem of generating inductive hypotheses that is based on ASP. Despite the interest in nonmonotonic ILP [26], there is a lack of available tools. [24], a nonmonotonic ILP system, does not provide the required support for the methodology presented in this paper. In particular, the system is not designed to tolerate noise in training examples and outputs only maximally compressive solutions.

6 Conclusions

We have presented an approach for rule learning in probabilistic nonmonotonic domains. The approach estimates probabilities for rules which, together with a knowledge base, minimise the error between a target probability and the entailed probability over a set of observed atoms. Such rules can be used to improve the original knowledge base, reducing the gap between the current knowledge and the observations, as we have shown in a planning scenario where executing the learned rules reduces the overall rate of the robotic agent failing to reach its goal.

NoMPRoL has a potentially broad applicability, ranging from situations involving interactions with the external environment, such as is the case in self-adaptive systems, to cases where a rich logical representation is integrated with uncertainty (e.g. in software engineering [1], in which nonmonotonic ILP is used to extend a partial system specification with learned requirements). Our approach would enable such systems to revise their knowledge bases with probabilistic information that is difficult or impossible for the designer to provide. Currently the main limit of the approach is scalability. As the space of candidate hypotheses and the domain entities and relations grows, the answer set solver has to deal with exponentially large ground theories. We plan further experiments aimed at improving the scalability of the methodology and extensions towards an incremental online learning setting.

References

1. Alrajeh, D., Ray, O., Russo, A., Uchitel, S.: Extracting Requirements from Scenarios with ILP. In: Muggleton, S.H., Otero, R., Tamaddoni-Nezhad, A. (eds.) ILP 2006. LNCS (LNAI), vol. 4455, pp. 64–78. Springer, Heidelberg (2007)
2. Boutilier, C., Brafman, R.I., Geib, C.: Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, pp. 1156–1162 (1997)
3. Clark, K.L.: Negation as failure. Logic and Data Bases, 293–322 (1977)
4. Corapi, D., Russo, A., Lupu, E.: Inductive logic programming as abductive search. In: Tec. Comm. of the 26th ICLP, LIPIcs, vol. 7, pp. 54–63, Dagstuhl (2010)
5. Dantsin, E.: Probabilistic logic programs and their semantics. In: Voronkov, A. (ed.) RCLP 1990 and RCLP 1991. LNCS, vol. 592, pp. 152–164. Springer, Heidelberg (1992)
6. De Raedt, L., Thomas, G., Getoor, L., Kersting, K., Muggleton, S. (eds.): Probabilistic, Logical and Relational Learning. Schloss Dagstuhl (2008)
7. De Raedt, L., Thon, I.: Probabilistic Rule Learning. In: Frasconi, P., Lisi, F.A. (eds.) ILP 2010. LNCS (LNAI), vol. 6489, pp. 47–58. Springer, Heidelberg (2011)
8. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning). The MIT Press, Cambridge (2007)
9. Giunchiglia, F., Traverso, P.: Planning as model checking. In: Biundo, S., Fox, M. (eds.) ECP 1999. LNCS, vol. 1809, pp. 1–20. Springer, Heidelberg (2000)
10. Gutmann, B., Kimmig, A., Kersting, K., De Raedt, L.: Parameter learning in probabilistic databases: A least squares approach. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part I. LNCS (LNAI), vol. 5211, pp. 473–488. Springer, Heidelberg (2008)
11. Inoue, K., Sato, T., Ishihata, M., Kameya, Y., Nabeshima, H.: Evaluating abductive hypotheses using an em algorithm on bdds. In: Boutilier, C. (ed.) IJCAI, pp. 810–815 (2009)
12. Inoue, K., Sakama, C.: Transforming abductive logic programs to disjunctive programs. In: Proc. 10th ICLP, pp. 335–353. MIT Press, Cambridge (1993)

13. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., De Raedt, L.: On the efficient execution of probLog programs. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 175–189. Springer, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-89982-2_22
14. Kok, S., Domingos, P.: Learning the structure of markov logic networks. In: Proceedings of the 22nd International Conference on Machine Learning, pp. 441–448. ACM Press, New York (2005)
15. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Gen. Comput. 4(1), 67–95 (1986)
16. Lavrac, N., Dzeroski, S.: Inductive logic programming - techniques and applications. Ellis Horwood series in artificial intelligence. Ellis Horwood (1994)
17. Lifschitz, V.: Answer set programming and plan generation. Artificial Intelligence 138(1-2) (2002); Knowledge Representation and Logic Programming
18. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Heidelberg (1987)
19. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
20. Muggleton, S.: Inverse entailment and progol. New Gen. Comp. 13(3&4), 245–286 (1995)
21. Muggleton, S.: Learning structure and parameters of stochastic logic programs. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 198–206. Springer, Heidelberg (2003)
22. Muggleton, S.: Learning stochastic logic programs. Electron. Trans. Artif. Intell. 4(B), 141–153 (2000)
23. Poole, D.: Abducing through negation as failure: stable models within the independent choice logic. The Journal of Logic Programming 44(1-3), 5–35 (2000)
24. Ray, O.: Nonmonotonic abductive inductive learning. Journal of Applied Logic 7(3), 329–340 (2009), http://www.cs.bris.ac.uk/Publications/Papers/2001069.pdf
25. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1-2), 107–136 (2006)
26. Sakama, C.: Nonmonotonic inductive logic programming. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, p. 62. Springer, Heidelberg (2001)
27. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: ICLP, pp. 715–729 (1995)
28. Schoppers, M.: Universal plans for reactive robots in unpredictable environments. In: IJCAI, vol. 87, pp. 1039–1046 (1987)
29. Shanahan, M.: An abductive event calculus planner. The Journal of Logic Programming 44(1-3), 207–240 (2000)
30. Singla, P., Domingos, P.: Discriminative training of markov logic networks. In: Veloso, M.M., Kambhampati, S. (eds.) AAAI. pp. 868–873. AAAI Press / The MIT Press (2005)
31. Zettlemoyer, L.S., Pasula, H., Kaelbling, L.P.: Learning planning rules in noisy stochastic worlds. In: AAAI, pp. 911–918 (2005)