Towards Structure Learning under the Credal Semantics

David Tuckey, Krysia Broda, and Alessandra Russo*

Imperial College London, UK
{dwt17, k.broda, a.russo}@ic.ac.uk

Abstract. We present the Credal-FOIL system for structure learning of probabilistic logic programs under the credal semantics. The credal semantics is a generalisation of the distribution semantics based on the answer set semantics. Our learning approach takes a set of examples that are atoms with target lower and upper bounds probabilities and a background knowledge that can have negative loops. We define accuracy in this setting and learn a set of normal rules without loops that maximises this notion of accuracy. We showcase the system on two proof-of-concept examples.

Keywords: Structure learning · Credal Semantics · Probabilistic Inductive Logic Programming

1 Introduction

Probabilistic Logic Programming is a field that has its roots in the nineties. It augments the field of Logic Programming to allow probabilistic inference using logical rules. While there exist many formulations, a major approach is based on the Distribution Semantics [22] (DS). In this context, a possible syntax consists in dividing probabilistic logic programs (PLP) into a logic program detailing deterministic relations and a set of independent probabilistic facts (events) to yield a single probability distribution over the atoms of the program (through a probability distribution over possible worlds). A well-known tool based on the DS is ProbLog [8] where programs are stratified.

The credal semantics [3] is a generalisation of the DS for unstratified probabilistic logic programs under the answer set semantics [10]. Informally, this semantics attributes to the atoms of a program a set of probability measures (instead of a single one) and by taking the extremums of this set, we obtain the notion of a lower-bound and upper-bound for the probability of each atom.

Rule learning is a well studied subject in logic programming and is often called inductive logic programming [17] (ILP). The goal is to find an hypothesis that, together with a background knowledge, covers a set of positive examples while not covering any of the negative examples. Its probabilistic variant is called probabilistic inductive logic programming [6, 21]. Another popular type of learning with PLP is parameter learning: with one set of rules, we aim to estimate the best probabilities of the initial events to describe the data [22, 1]. Some systems do both types of learning at the same time [5, 2].

In this paper we introduce Credal-FOIL, a framework and algorithm for learning the structure of probabilistic logic programs under the credal semantics. In Credal-FOIL the learned hypothesis is a non-probabilistic set of normal rules with no negative loops and the logic program in the background knowledge is expressed as a non stratified answer set program. Examples in our Credal-FOIL learning task are single atoms with an associated target lower and upper bound probability. To the best of our knowledge no attempt had been proposed yet for structure learning under the credal semantics. Our system defines a notion of coverage over our examples and learns incrementally a hypothesis that maximises a notion of accuracy. We showcase our system using two proof of concept examples: the first example shows that in the specific case when the background knowledge is stratified the learning task coincides with the ProbFOIL [7] learning task. In the second example we show that our algorithm allows also for a background knowledge that is not stratified so exploiting in full the credal semantics.

The rest of the paper is composed as follows: Section 2 presents the background necessary for reading this paper, Section 3 presents our learning task and our methodology to solve it. We apply our system to two examples in Section 4 and talk about related works in Section 5. Finally we discuss our work and next steps in Section 6 and conclude in Section 7.

2 Background

This section introduces notions of logic programming [14] and answer set semantics [10] to then present the credal semantics [4].

2.1 The answer set semantics

In a logic program, a term is either a constant $a, b, c..$ or a variable $X, Y, Z..$, an atom is of the form $p(t_1, ..., t_n)$ where p is a predicate of arity n and $t_1, ..., t_n$ are terms (n can be 0). A literal is an atom p or a negated atom *not* p (*negation as failure*), a *normal* rule is of the form A :- $B_1, ..., B_n$ where A is an atom (called the *head* of the rule) and $B_1, ..., B_n$ are literals (called the *body* of the rule). A rule with an empty body is called a fact. A *ground* literal is a literal with no variable, a *ground* rule contains only ground literals. A normal logic program is a set of normal rules. The Herbrand universe of a program is the set of all the ground terms appearing in the program. The Herbrand Base HB_P of a program P is the set of all the ground atoms that can be formed from the predicates in P and the ground terms in the Herbrand universe. A *substitution* θ is a mapping from variables to terms. The *grounding* of a program P is the set of all ground rules $r\theta$ where r is in P and θ is a mapping from variables to ground terms in the Herbrand universe. We call an *interpretation* of P any subset of HB_P.

We can now introduce the answer set semantics. Given a program P and an interpretation I, the reduct P^I is the program made from the grounding of P in the following way: remove all rules that has in the body the negation of an atom in I and then delete all negative literals from the body of the remaining rules. The reduct P^I is a ground definite program and thus has a single minimal model. I is an *answer set* of P if it is the minimal model of P^I. We denote the set of answer sets of P as $AS(P)$. A *stratified* program P is a program that can be divided in the form $P = P_1 \cup P_2 \cup ... \cup P_m$ where P_i and P_j are disjoint for all $i \neq j$ such that if an atom occurs positively in P_i then it appears in the head of a rule in P_j with $j \leq i$ and if an atom appears negatively in P_i then it appears in the head of a rule in P_j with $j < i$. It is known that a stratified program admits one single answer set if any, and that unstratified programs can admit more than one. As a program may have multiple answer sets, the entailment of an atom can be of two types: brave and cautious. An atom p is *cautiously* entailed by P if p is in all the answers sets of P and *bravely* entailed by P if p is in at least one answer set of P. For an interpretation I and conjunctive formula F, we denote $I \models F$ if F is entailed (true) in I. The notion of brave and cautious entailment is extended to conjunctive formulas.

2.2 The credal semantics

The credal semantics is a generalisation of the Distribution semantics [22]: it uses the same interpretation for probabilistic facts given in the PLP but attributes to it a set of probability measures. For an extensive review of probabilistic logic programming we redirect the reader to [20] and for a more in-depth description of the credal semantics to [15, 3, 4].

The syntax we consider in this paper for probabilistic logic programs is as follows: a PLP is a pair $< P, PF >$ where P is a logic program and PF is a set of independent ground probabilistic facts. A probabilistic fact is of the form $\alpha :: p$ (using the Problog notation [8]) where p is an atom (of any arity) and $\alpha \in [0, 1]$ is the probability with which p is true. We say that a PLP is stratified if P is a stratified logic program. Let n be the cardinality of PF. Informally, we can choose a probabilistic fact $\alpha :: p$ with probability α and discard it with probability $1 - \alpha$. A *total choice* C is a subset of PF in which we consider to have "chosen" p if $p \in C$ (p is chosen true) and "discarded" p if $p \notin C$ (p is chosen false). There exist 2^n total choices. To each total choice C we associate the probability $Prob(C) = \prod_{\alpha::p \in C} \alpha$ (as the probabilistic facts are independent). Let's define $PF_C = \{p | \alpha :: p \in C\}$ to be the set of facts that correspond to the total choice C and $\wedge C = \wedge_{\alpha::p \in C}\ p$ the formula corresponding to the conjunction of all the facts in C.

We denote HB_P^* the Herbrand Base HB_P augmented with the atoms in the probabilistic facts. When referring to interpretations of P in the context of a PLP $< P, PF >$, we mean subsets of HB_P^*. Probabilities are defined over interpretations of P using probability measures Pr: for an interpretation $I \subseteq$

```
0.4::sunshine(d1).
0.8::warm(d1).
0.3::tired(d1).
0.2::wind(d1).
var1(d1).
run(X) :- var1(X), not walk(X), sunshine(X), not wind(X).
walk(X) :- var1(X), not run(X).
sick(X) :- var1(X), run(X), tired(X).
sick(X) :- var1(X), walk(X), not warm(X).
```

Fig. 1. Probabilistic logic program with a negative loop. It can be sunny, warm or windy and I can be tired on day 1. I can always go for a walk, and I can only go for a run if it is sunny and not windy. Obviously, I cannot go for a run and a walk at the same time. Note that I can decide to go for a walk even if it is sunny and not windy. If I go for a run when I'm tired or if I go for a walk and it is not warm then I get sick

HB_P^* and a probability measure Pr over $2^{HB_P^*}$, $Pr(I)$ denotes the probability associated to I by Pr. We surcharge the notation and define the probability of a conjuctive formula F as $Pr(F) = \sum_{I \in 2^{HB_P}, I \models F} Pr(I)$ for a given probability measure Pr over $2^{HB_P^*}$. A PLP is *consistent* if there is at least one answer set for each total choice, meaning that $\forall C \in 2^{PF}, AS(P \cup PF_C) \neq \emptyset$. We give the definition from Cozman and Mauá [3]: a *probability model* for a consistent PLP $< P, PF >$ is a probability measure Pr over interpretations of P such as:

- every interpretation $I \subseteq HB_P^*$ with $Pr(I) > 0$ has to be an answer set for a total choice: $Pr(I) > 0 \Rightarrow \exists C \in 2^{PF}$ such that $I \in AS(P \cup PF_C)$
- for C total choice: $Pr(\wedge C) = Prob(C)$

The credal semantics of a PLP is the set of all its probability models. In other words, we accept all probability measures that only give non-zero probability to elements of $\{AS(P \cup PF_C) | C \text{ total choice}\}$ such that for each total choice, the sum of the probability of the corresponding answer sets equals the probability of the total choice. Now, the probability of a conjunctive formula F can be different depending on the probability model considered. However there exists a minimum value [3] $\underline{Pr}(F)$ to this probability: whatever the probability model Pr considered we have $\underline{Pr}(F) \leq Pr(F)$. Similarly there exists a maximum value $\overline{Pr}(F)$ to this probability: whatever the probability model Pr considered we have $\overline{Pr}(F) \geq Pr(F)$. We call $\underline{Pr}(F)$ (resp. $\overline{Pr}(F)$) the lower bound probability of F (resp. upper bound probability of F). Naturally, it always holds that $\underline{Pr}(F) \leq \overline{Pr}(F)$.

An algorithm is given in [3] for computing the lower and upper bound probability using the answer sets of $P \cup PF_C$. Informally, $\underline{Pr}(F)$ (resp. $\overline{Pr}(F)$) is the sum of the probabilities of the total choices $Prob(C)$ where F is cautiously (resp. bravely) entailed by $P \cup PF_C$.

We give an example of probabilistic logic program in Figure 1. It defines 4 probabilistic facts with their corresponding probabilities, which yield $2^4 = 16$

total choices. There will be a negative loop in $P \cup PF_C$ for all the total choices C where *sunshine* is true and *wind* is false at the same time (that is with probability $0.4 * 0.8 = 0.32$), so *run(d1)* appears in one out of two answer sets for these total choices. Otherwise *run* is always false. That makes for: $\underline{Pr}(run(d1)) = 0$ and $\overline{Pr}(run(d1)) = 0.32$. While the atom *sick(d1)* is not in a negative loop, it depends on *run(d1)* or *walk(d1)* and so will have different lower and upper bound probabilities: $\underline{Pr}(sick(d1)) = 0.15$ and $\overline{Pr}(sick(d1)) = 0.27$.

3 Structure learning under the credal semantics

The first difficulty of learning a probabilistic program under the credal semantics is that we aim to learn a set of probability distributions instead of a single one. We use as targets the lower and upper bound probabilities of example atoms. Our approach builds on ProbFOIL [7] to adapt inductive logic programming paradigms to our setting. Calling the hypothesis space S_M, we define our learning task:

Definition: Learning task. Given a set of examples $E = \{(e_i, \underline{p}_i, \overline{p}_i) | i = 1, .., n\}$ where each e_i are grounded atoms, $\underline{p}_i$ (resp. $\overline{p}_i\}$) is the target lower (resp. upper) bound probability for e_i and a background knowledge $B =< P, PF >$ in the form of a (possibly non-stratified) PLP, find an hypothesis $H \in S_M$ that maximises accuracy (as defined later in Equation 3).

The hypothesis we learn here is a set of rules with no negative loops. In practice, these examples are different groundings of the same predicate (the target predicate) and the rules in the hypothesis all have the same head. We extend ProbFOIL [7] to define the coverage of our probabilistic examples and notions such as accuracy, precision and recall.

Each example can be interpreted in the following way: they provide information about the proportion of the worlds (total choices) in which the atom e_i should be true or false, the probability of each total choice being given by the DS. In this context, we interpret the probabilities as percentages and when talking about proportions of the worlds, we mean taking into account the probabilities of each world. For example when writing “in 0.5 of the worlds”, we do not mean in exactly half of them but in a certain amount such that their probabilities sum to 0.5. Now as in our setting one total choice can lead to multiple answer sets, we not only have atoms being true or false but also the notion of brave and cautious inference for each total choice. Each example provides targets for these different inference types. Informally, $(e_i, \underline{p}_i, \overline{p}_i)$ can be interpreted as meaning that e_i should be cautiously entailed (cautiously true) in $\underline{p}_i$ (in percentage) of the worlds, bravely entailed (bravely true) in $(\overline{p}_i - \underline{p}_i)$ of the worlds and not entailed (false) in $(1 - \overline{p}_i)$ of the worlds. We call respectively these quantities the “True/-Positive” $(t_i = \underline{p}_i)$, “Brave” $(b_i = \overline{p}_i - \underline{p}_i)$ and “False/Negative” $(f_i = 1 - \overline{p}_i)$ parts of example $(e_i, \underline{p}_i, \overline{p}_i)$. This is illustrated in Figure 2. We can then define the *Positive*, *Brave* and *Negative* parts of the dataset as:

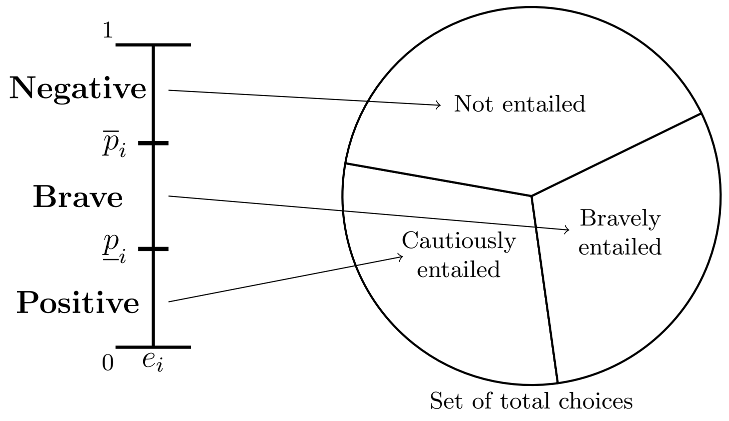

Fig. 2. Example $(e_i, \underline{p}_i, \overline{p}_i)$ can be divided into three parts: positive, brave and negative. Respectively, each part represents in how many total choices (in percentage) the example atom should be cautiously entailed, bravely entailed and not entailed.

$$T = \sum_{i=1..n} t_i \qquad B = \sum_{i=1..n} b_i \qquad N = \sum_{i=1..n} f_i \tag{1}$$

To rate an hypothesis, we use the notions of ROC analysis (True Positive, False Positive, True Negative, False Negative) to which we introduce two new categories: True Brave (TB) and False Brave (FB). These are aimed at defining the coverage over the brave part of the dataset. It is important to note that each example participates at the same time to the Positive, Brave and Negative parts of the dataset. Given the PLP $B \cup H =< P \cup H, PF >$ composed of background $B =< P, PF >$ together with an hypothesis H and an example $(e_i, \underline{p}_i, \overline{p}_i)$, let $\underline{q}_i = \underline{Pr}_{B\cup H}(e_i)$ and $\overline{q}_i = \overline{Pr}_{B\cup H}(e_i)$ the predicted lower probability and predicted upper probability for atom e_i. The aforementioned quantities are defined on each example $(e_i, \underline{p}_i, \overline{p}_i)$ as follows:

– true positive $tp_i = min(\underline{p}_i, \underline{q}_i)$
– true negative $tn_i = min(1 - \overline{p}_i, 1 - \overline{q}_i)$
– true brave $tb_i = max(0, min(\overline{p}_i, \overline{q}_i) - max(\underline{p}_i, \underline{q}_i))$
– false positive $fp_i = max(0, \underline{q}_i - \underline{p}_i)$
– false negative $fn_i = max(0, \overline{p}_i - \overline{q}_i)$
– false brave left $fbl_i = max(0, min(\underline{p}_i, \overline{q}_i) - \underline{q}_i)$
– false brave right $fbr_i = max(0, \overline{q}_i - max(\underline{q}_i, \overline{p}_i))$

These quantities are illustrated in Figure 3. If $\underline{q}_i < \underline{p}_i$ (ex. 1,2 and 5 of Figure 3), that means that in $\underline{q}_i$ of the worlds, we rightfully predict that e_i is cautiously entailed so $tp_i = \underline{q}_i$. However if $\underline{q}_i > \underline{p}_i$ (ex. 3,4 and 6 of Figure 3), we predict that e_i is cautiously entailed in $\underline{q}_i$ of the worlds, which is "too many". Thus $tp_i = \underline{p}_i$ and $fp_i = \underline{q}_i - \underline{p}_i$, meaning that we predict that in fp_i worlds e_i is

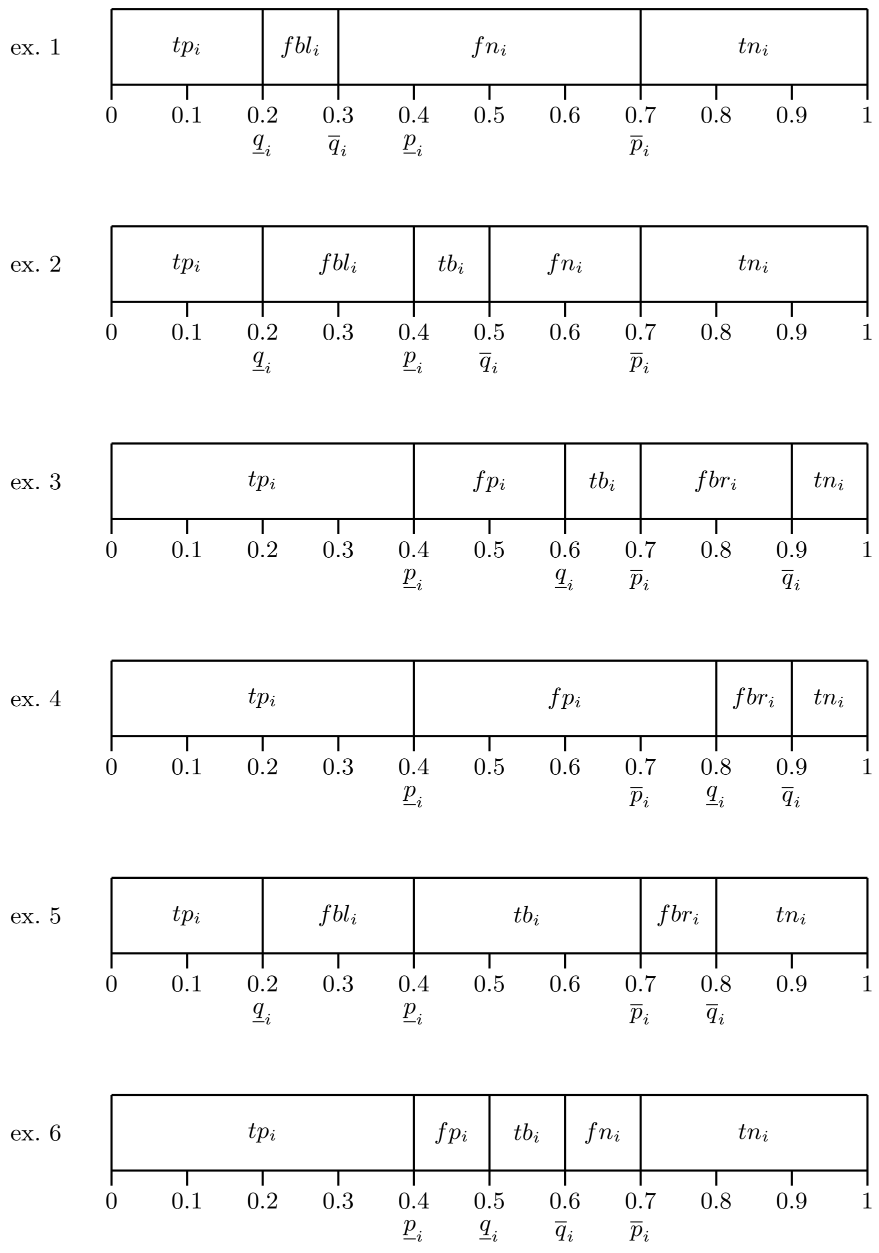
ex. 1
tp_i fbl_i fn_i tn_i
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
$\underline{q}_i$ $\overline{q}_i$ $\underline{p}_i$ $\overline{p}_i$
ex. 2
tp_i fbl_i tb_i fn_i tn_i
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
$\underline{q}_i$ $\underline{p}_i$ $\overline{q}_i$ $\overline{p}_i$
ex. 3
tp_i fp_i tb_i fbr_i tn_i
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
$\underline{p}_i$ $\underline{q}_i$ $\overline{p}_i$ $\overline{q}_i$
ex. 4
tp_i fp_i fbr_i tn_i
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
$\underline{p}_i$ $\overline{p}_i$ $\underline{q}_i$ $\overline{q}_i$
ex. 5
tp_i fbl_i tb_i fbr_i tn_i
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
$\underline{q}_i$ $\underline{p}_i$ $\overline{p}_i$ $\overline{q}_i$
ex. 6
tp_i fp_i tb_i fn_i tn_i
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
$\underline{p}_i$ $\underline{q}_i$ $\overline{q}_i$ $\overline{p}_i$

Fig. 3. Graphical representation of all the 6 possible arrangements of predicted $(\underline{q}_i, \overline{q}_i)$ and target $(\underline{p}_i, \overline{p}_i)$ low and upper bound probabilities together with the associated tp_i, tb_i, tn_i, fp_i, fn_i, fbl_i, fbr_i quantities. The target lower $\underline{p}_i$ and upper $\overline{p}_i$ probabilities are unchanged between the examples. Let's call $[\underline{q}_i, \overline{q}_i]$ the predicted segment and $[\underline{p}_i, \overline{p}_i]$ the target segment. Example 1 is the case when the predicted segment is "to the left" of the target segment and does not intersect. Example 2 is the case when the predicted segment "overflows" to the left of the target segment. Example 3 is the overflow to the right. Example 4 is when the predicted segment is to the right of the target segment without intersecting. Example 5 shows the case when the predicted segment contains the target segment and example 6 the case when the predicted segment is contained in the target segment.

cautiously entailed when it should not be. The argument is of the same nature for tn_i and fn_i depending on $\overline{p}_i$ and $\overline{q}_i$. Now the true brave part of the example is the intersection of the two segments $[\underline{p}_i, \overline{p}_i]$ and $[\underline{q}_i, \overline{q}_i]$. There are two "ways of obtaining" some false brave: the segment $[\underline{q}_i, \overline{q}_i]$ overflows $[\underline{p}_i, \overline{p}_i]$ to the left or to the right. When it overflows to the left then $\underline{q}_i < \underline{p}_i$ (assume $\overline{q}_i > \underline{p}_i$, ex. 2 and 5 of Figure 3), then we predict that in $fbl_i = \underline{p}_i - \underline{q}_i$ of the worlds e_i is bravely entailed when it should be cautiously entailed. Symmetrically, when $[\underline{q}_i, \overline{q}_i]$ overflows to the right, $\overline{q}_i > \overline{p}_i$ (assume $\underline{q}_i < \overline{p}_i$, ex. 3 and 5 of Figure 3), we predict that in $fbr_i = \overline{q}_i - \overline{p}_i$ of the worlds e_i is bravely entailed when it should no be entailed. Informally, *false brave left* can be seen as "we predict bravely entailed when it should be cautiously entailed" and *false brave right* as "we predict bravely entailed when it should not be entailed". The same arguments hold when the segments $[\underline{p}_i, \overline{p}_i]$ and $[\underline{q}_i, \overline{q}_i]$ do not intersect (ex. 1 and 4 of Figure 3). We compute these quantities over the entire dataset by summing them over each individual example:

$$\begin{aligned} TP &= \sum_{i=1..n} tp_i & TB &= \sum_{i=1..n} tb_i \\ TN &= \sum_{i=1..n} tn_i & FP &= \sum_{i=1..n} fp_i \\ FN &= \sum_{i=1..n} fn_i & FBR &= \sum_{i=1..n} fbr_i \\ FBL &= \sum_{i=1..n} fbl_i & FB &= FBL + FBR \end{aligned} \tag{2}$$

The false brave (FB) quantity is the sum of the false brave left (FBL) and the false brave right (FBR). These allow us to define the following loss functions:

$$\begin{aligned} accuracy &= \frac{TP + TN + TB}{TP + TN + TB + FN + FP + FB} \\ precision &= \frac{TP}{TP + FP + FBR} \\ recall &= \frac{TP}{TP + FN + FBL} \\ \text{m-estimate} &= \frac{TP + m * \dfrac{P}{P + N + B}}{TP + FP + FBR + m} \\ \text{m-estimate}_{tb} &= \frac{TP + TB + m * \dfrac{P + B}{P + N + B}}{TP + TB + FP + FBR + m} \end{aligned} \tag{3}$$

They are a modification of the usual heuristics from ROC analysis used in inductive logic programming [9]. For example *precision* usually represents the ratio of true positives over all predicted positive examples. We add FBR to the denominator to penalise an hypothesis that is also “too bravely optimistic”, while not adding TB to the numerator to get the precision to prioritise keeping $\underline{q}_i$ under $\underline{p}_i$. We defined m-estimate$_{tb}$ to favor hypotheses with better TP and TB at the same time. In the case of a stratified background knowledge we get $\underline{q}_i = \overline{q}_i$ and $\underline{p}_i = \overline{p}_i$ which makes that $TB = FBR = FBL = 0$. Thus our loss functions reduce to the usual ones and the other quantities (TB, TN, FP, FN) are equivalent to their definition in ProbFOIL [7].

Our algorithm (Figure 4) is a version of the coverage algorithm and is similar to that of ProbFOIL+'s [5]. The algorithm is composed of an outer loop which iteratively adds a rule to the hypothesis. The outer loop stops once adding a new rule does not improve the accuracy. At each iteration, it finds a most specific rule (line 6 of Algorithm 1) that is then pruned (line 7) to get the sub-rule that maximizes the localscore:

$$localscore(H, c) = \text{m-estimate}_{tb}(H \cup \{c\}) - \text{m-estimate}_{tb}(H) \qquad (4)$$

This pruning checks for which i the rules $h \leftarrow b_1, .., b_i$ maximises this localscore. As such, multiple (or none or all) literals might be removed from the body. The refinement process is shown in Algorithm 2: from a rule with empty body, we iteratively refine it until either it does not increase the TP or TB part of the dataset or does not have any FP or FBR (line 2). The first step to the refinement loop consists of finding the set of all possible literals that can be added at the end of the rule r (line 3). Next we reject some possible refinements (line 4) to compensate for the fact that we do not discard covered examples as a normal FOIL algorithm would. In our case, we cannot discard the examples as a rule might only cover a part of that example. This implies that when learning a second rule, we need to make sure it is not going to cover exactly the same part of the examples as the first one. The reject function checks each possible refinement (literal) by appending it to the body of the rule r and checking if adding this latter to the hypothesis changes the computed probabilities. If the probabilities stay the same, that means that this potential rule would not be useful, thus the refinement is rejected. Finally, the refinement process selects the best literal in the set of retained refinements (lines 7-8) and appends it to the current rule. This process is repeated until the stopping criteria is met.

In practice, the refinement process of Algorithm 2 is included within a beam search to allow for a better search through the hypothesis space. We keep the rules that had the best maximum localscore until none can be refined further, at which point we return the one with the maximum potential localscore.

4 Proof of concept

We demonstrate our system Credal-FOIL on two examples of probabilistic program reconstruction and then give some implementation details about our experiments. The first is the surfing example from De Raedt et Thon [7] while the second is the program in Figure 1 which has a negative loop in the background knowledge. To define the hypothesis space, we use a simple mode declaration [16]:

Algorithm 1: Covering algorithm

```
H = ∅
h = target_predicate
Let $S_M$ be the hypothesis space
stop_search = False
While not stop_search:
    Let (h ← b$_1$, .., b$_n$) be find_best_rule($S_M$, H, h)
    new_rule = $argmax_i$ localscore(H, h←b$_1$,..,b$_i$) % i ∈ [1,n]
    if accuracy(H) ≥ accuracy(H ∪ {new_rule}):
        stop_search=True
    else:
        H = H ∪ {new_rule}
return H
```

Algorithm 2: find_best_rule(S_M, H, h)

```
r = h ←
While not (TP(H∪{r})-TP(H)+TB(H∪{r})-TB(H)) == 0 OR (FP({r}) +
     FBR({r})) == 0):
    refinements = refinement($S_M$, r) %Set of all possible
         literals that can be added to c based on $S_M$
    refinements = reject(H, r, refinements) %We reject some
         refinements
    Let (h ← b$_1$, .., b$_i$) be r % i ≥ 0
    Let N be the cardinality of refinements
    l = argmax$_{k\in[1,N]}$ localscore(H, h ← b$_1$,..,b$_i$,refinements[k])
    r = h ← b$_1$,..,b$_i$,refinements[l]
return r
```

Fig. 4. Listing of the algorithm of our system Credal-FOIL.

modeh(pred1(Var)) (mode head) and *modeb(pred2(Var))* (mode body) respectively define that "pred1(X)" can be in the head of a rule and that "pred2(X)" and "not pred2(X)" can be in the body of a rule. In our examples, we only use predicates of arity one. Of course with our learning setting, there can only be one mode head declaration.

In the following examples, an added atom *var1(X)* appears in the programs. It is automatically added by our learning system for practical reasons but does not influence the probabilities. We explain why we do so in section 4.3.

```
0.03::pop(t1).
0.67::windok(t1).
0.6::sunshine(t1).
var1(t1).
surfing(X) :- var1(X), not pop(X), windok(X).
surfing(X) :- var1(X), not pop(X), sunshine(X).
```

Fig. 5. Full program of the surfing example from De Raedt et Thon [7]. The goal is to learn the last two rules.

4.1 Surfing example

The *surfing* example has three probabilistic facts per example: *pop*(t_i), *windok*(t_i) and *sunshine*(t_i) which each have a probability. The full program is given in Figure 5. The mode bias allows only for *surfing* in the head and for *pop*, *windok* and *sunshine* (and their negation by failure) in the body. We generated 50 different examples by grounding each set of three probabilistic facts 50 times and attributing to each grounding a random probability. We then computed the probability of *surfing*(t_i) (where $i = 1..50$) using the full program to obtain a set of examples of the form: *e(surfing*(t_1)*; 0.84; 0.84)* (which correspond to probabilities 0.03, 0.67 and 0.6 to *pop*(t_1), *windok*(t_1) and *sunshine*(t_1) respectively). We can note here that as the PLP is stratified, the target upper and lower probabilities are the same. Using the value $m = 5$ for the m-estimate$_{tb}$, the program terminated after 3 iterations of the outer-loop with the following rules:

surfing(X) :- var1(X), not pop(X), windok(X).
surfing(X) :- var1(X), sunshine(X), not pop(X).
surfing(X) :- var1(X).

The third rule was rejected by Credal-FOIL as it lowered the overall accuracy. This example aims to show that our system works in the case of point probabilities and is not only tailored for unstratifed background knowledge.

4.2 Sickness example

We now consider the program in Figure 1 and use as target predicate *sick*. The mode declaration allows for *tired*, *run*, *walk* and *warm* (and their negation by failure) in the body. We proceed in the same way as with the *surfing* example to compute (using the full program) the examples of the form: *e(sick(t1); 0.15; 0.27)*. For learning, we delete from the program the two rules with *sick* in the head. The logic program part of the background knowledge thus contains the two rules with *run* and *walk* in the head, forming a negative loop. We attempt to reconstruct the original program using 50 examples (*sick(ti)* for $i = 1..50$). Using the value $m = 5$ for the m-estimate$_{tb}$, after three outer-loop iterations we obtain the following rules:

sick(X) :- var1(X), not warm(X), not run(X).
sick(X) :- var1(X), run(X), tired(X).
sick(X) :- var1(X).

The third rule is of course discarded by the algorithm as it lowers the accuracy. As *not run(X)* and *walk(X)* are logically equivalent in this PLP, Credal-FOIL is capable here to recover the two missing rules.

4.3 Comments on the proof of concept

We give here some additional information about our experimental setting. Our system does not use knowledge compilation to compute the probability values but instead simply computes the answer sets for each total choice using CLINGO [23]. This is highly inefficient so, to make it tractable, we run every example separately. It makes that instead of having 2^{150} total choices, we only have $50 * 2^3$. We also paralellise the calls to CLINGO to speed up execution. The choice of using 50 examples was made to keep the run-time relatively low. In average, running time for the *sickness* task is of 8 and 14 minutes for 30 and 50 examples respectively. For the *surfing* example running time is of 5 and 7 minutes for 30 and 50 examples respectively. Experiments were run on a Intel i7-7700K 4.20GHz.

In fact, the *sickness* learning only needed two examples to find the original rules while *surfing* needed five examples at minimum. The factor that explains this difference is probably that m-estimate$_{tb}$ had access to more information (TB, FBR) in the *sickness* case.

CLINGO uses GRINGO to ground programs. Thus rules need to be safe, meaning amongst other things that negative literals cannot appear alone in the body of rules. To avoid this problem altogether, we automatically add to the body of rules the atom *var1(X)* and add ground facts *var1(ti)* ($i = 1..50$) in the program.

5 Related Works

Structure learning is not new in probabilistic logic programming and comes from the field of inductive logic programming. We took inspiration from ProbFOIL [7] which learns the same types of hypothesis under the DS (with a stratified background PLP). This system has been extended to simultaneously learn the structure and probability values in ProbFOIL+ [5]. One recent system is SLIPCOVER [2] that allows to learn multi-head rules together with probabilities. While ProbFOIL uses an approach similar to the one we presented, SLIPCOVER first generates a set of best rules and then inserts them one by one in the theory if it improves its score. SLIPCOVER is also capable of learning rules that have in the head a non target predicate. An interesting point is that each system has a different learning task: we maximise our definition of accuracy, ProbFOIL defines a regression problem and SLIPCOVER maximizes the log-likelihood of its examples.

While (to the best of our knowledge) our work is the first attempt at structure learning under the credal semantics, previous work has been done in learning using semantics that are based on the answer set semantics. PrASP [18] is such a system allowing definition of rules in first-order logic and probability intervals. The weight learning as described in Nickles and Mileo [19] aims at maximising the likelihood of the set of examples and is defined in the case of point probabilities. Another system called LP^{MLN} [12], bringing together the answer set semantics with markov-logic-networks, also allows for weight learning [13] with point probabilities.

6 Discussion

We discuss in this section our work and future steps. We will first comment on practical sides of our approach. We would like to point out that different parts of our system are based on empirical choices. The first is the choice of m-estimate$_{tb}$ in the local score: it has shown to perform better than the other candidates for unstratified PLPs. Also, the construction of our heuristic functions aimed to follow the original goals of each loss function in ROC analysis, but one might decide to use entirely new metrics based on the new categories we defined. Finally, our way to formulate our TB and FB quantities can be debatable. Consider the case where, for an example e_i, we have $\underline{p}_i = 0.5$, $\overline{p}_i = 0.7$, $\underline{q}_i = 0.1$ and $\overline{q}_i = 0.3$. Our system would consider that $fbl_i = 0.2$ and $fn_i = 0.4$ but one can argue it to be unfair as $\overline{p}_i - \underline{p}_i = \overline{q}_i - \underline{q}_i$, which means that we predict the example to be bravely entailed in the right amount of worlds. We could instead compute the quantities using the following formulas:

$$\begin{aligned} tb_i &= min(\overline{p}_i - \underline{p}_i, \overline{q}_i - \underline{q}_i) \\ fbl_i &= 0 \\ fbr_i &= \overline{q}_i - \underline{q}_i - tb_i \end{aligned} \tag{5}$$

We can see it as favoring the intervals $[\underline{p}_i, \overline{p}_i]$ and $[\underline{q}_i, \overline{q}_i]$ to be of the same lengths instead of intersecting. Empirically this formulation did not make much of a difference in our tests.

We made the choice of maximising our notion of accuracy which is intuitive, but a case could be made to instead formulate the learning task using a regression loss of the form (find H such as):

$$H = argmin_H \sum_{(e_i, \underline{p}_i, \overline{p}_i) \in E} |\underline{Pr}_{B \cup H}(e_i) - \underline{p}_i| + |\overline{Pr}_{B \cup H}(e_i) - \overline{p}_i| \tag{6}$$

This would require the definition of a specific heuristic function as it is not equivalent to maximising accuracy because of specific cases illustrated by examples 1 and 4 in Figure 3.

While we fulfilled our goal of proving the feasibility of learning under the credal semantics, expanding it in a concrete context will be complex. The main difficulty comes from our definition of examples: we required the target lower and upper bound probabilities. We do not have at the time a protocol to extract them from statistical data. While artificial intelligence has used semantics based on intervals (Dempster-Shafer Belief Functions [11] for example), they differ a lot in term of interpretation. As such more in-depth studies are necessary to understand how to extract from statistical data the lower and upper bounds probability for them to correspond to the meaning given by the credal semantics. Immediate future work will thus look into more applicable learning tasks which will require to define a precise context for the interpretation of the credal semantics. We also aim to add weight learning to our algorithm and will consider allowing negative loops in the hypothesis.

7 Conclusion

In this paper we presented the system Credal-FOIL aimed at learning the structure of a probabilistic logic program interpreted under the credal semantics. Credal-FOIL is capable of learning a set of normal rules with a single head using an unstratified background knowledge. From the set of examples and an hypothesis, we introduce heuristic functions that are inspired by ROC analysis. We make a proof-of-concept using two examples of program reconstruction using possibly non-stratified background knowledge. Future work will focus on learning weights and negative loops as well as considering alternative learning tasks.

References

1. Bellodi, E., Riguzzi, F.: Expectation maximization over binary decision diagrams for probabilistic logic programs. Intelligent Data Analysis **17**(2), 343–363 (2013). https://doi.org/10.3233/IDA-130582
2. Bellodi, E., Riguzzi, F.: Structure learning of probabilistic logic programs by searching the clause space. Theory and Practice of Logic Programming **15**(2), 169–212 (2015). https://doi.org/10.1017/S1471068413000689
3. Cozman, F.G., Mauá, D.D.: On the Semantics and Complexity of Probabilistic Logic Programs. CoRR **abs/1701.0** (2017), http://arxiv.org/abs/1701.09000
4. Cozman, F.G., Mauá, D.D.: The joy of Probabilistic Answer Set Programming: Semantics, complexity, expressivity, inference. International Journal of Approximate Reasoning pp. 91–101 (jul 2020). https://doi.org/10.1016/j.ijar.2020.07.004, https://linkinghub.elsevier.com/retrieve/pii/S0888613X20302012
5. De Raedt, L., Dries, A., Thon, I., Van Den Broeck, G., Verbeke, M.: Inducing probabilistic relational rules from probabilistic examples. IJCAI International Joint Conference on Artificial Intelligence **2015-Janua**, 1835–1843 (2015)
6. De Raedt, L., Kersting, K.: Probabilistic Inductive Logic Programming. In: Ben-David, S., Case, J., Maruoka, A. (eds.) Algorithmic Learning Theory. pp. 19–36. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

7. De Raedt, L., Thon, I.: Probabilistic rule learning. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **6489 LNAI**, 47–58 (2011)
8. Fierens, D., den Broeck, G.V., Renkens, J., Shterionov, D.S., Gutmann, B., Thon, I., Janssens, G., Raedt, L.D.: Inference and learning in probabilistic logic programs using weighted Boolean formulas. CoRR **abs/1304.6** (2013), http://arxiv.org/abs/1304.6810
9. Fürnkranz, J., Flach, P.A.: ROC n' rule learning - Towards a better understanding of covering algorithms. Machine Learning **58**(1), 39–77 (2005). https://doi.org/10.1007/s10994-005-5011-x
10. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. 5th International Conf. of Symp. on Logic Programming (December), 1070–1080 (1988). https://doi.org/10.1.1.24.6050
11. Halpern, J.Y.: Reasoning about Uncertainty (2018). https://doi.org/10.7551/mitpress/10951.001.0001
12. Lee, J., Wang, Y.: A probabilistic extension of the stable model semantics. In: Logical Formalizations of Commonsense Reasoning - Papers from the AAAI Spring Symposium, Technical Report. vol. SS-15-04, pp. 96–102. AI Access Foundation (2015)
13. Lee, J., Wang, Y.: Weight learning in a probabilistic extension of answer set programs. Principles of Knowledge Representation and Reasoning: Proceedings of the 16th International Conference, KR 2018 pp. 22–31 (2018)
14. Lloyd, J.W.: Foundations of logic programming (1984)
15. Lukasiewicz, T.: Probabilistic Description Logic Programs. In: Godo, L. (ed.) Symbolic and Quantitative Approaches to Reasoning with Uncertainty. pp. 737–749. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
16. Muggleton, S.: Inverse entailment and progol. New Generation Computing **13**(3), 245–286 (1995). https://doi.org/10.1007/BF03037227, https://doi.org/10.1007/BF03037227
17. Muggleton, S., de Raedt, L.: Inductive Logic Programming: Theory and methods. The Journal of Logic Programming **19-20**(2), 629–679 (may 1994). https://doi.org/10.1016/0743-1066(94)90035-3, https://linkinghub.elsevier.com/retrieve/pii/0743106694900353
18. Nickles, M.: A Tool for Probabilistic Reasoning Based on Logic Programming and First-Order Theories Under Stable Model Semantics. In: Michael, L., Kakas, A. (eds.) Logics in Artificial Intelligence. pp. 369–384. Springer International Publishing, Cham (2016)
19. Nickles, M., Mileo, A.: A System for Probabilistic Inductive Answer Set Programming. Lecture Notes in Computer Science **9310**, 99–105 (2015)
20. Riguzzi, F.: Foundations of Probabilistic Logic Programming. River Publishers (2018)
21. Riguzzi, F., Bellodi, E., Zese, R.: A History of Probabilistic Inductive Logic Programming. Frontiers in Robotics and AI **1**, 6 (2014). https://doi.org/10.3389/frobt.2014.00006, https://www.frontiersin.org/article/10.3389/frobt.2014.00006
22. Sato, T.: A Statistical Learning Method for Logic Programs with Distribution Semantics. In: ICLP (1995)
23. University of Potsdam: Potsdam answer set solving collection, https://potassco.org/