Inducing Probabilistic Relational Rules from Probabilistic Examples*

Luc De Raedt, Anton Dries, Ingo Thon[†], Guy Van den Broeck, Mathias Verbeke[‡]
KU Leuven, Department of Computer Science
Celestijnenlaan 200A, BE-3001 Heverlee, Belgium

Abstract

We study the problem of inducing logic programs in a probabilistic setting, in which both the example descriptions and their classification can be probabilistic. The setting is incorporated in the probabilistic rule learner ProbFOIL$^+$, which combines principles of the rule learner FOIL with ProbLog, a probabilistic Prolog. We illustrate the approach by applying it to the knowledge base of NELL, the Never-Ending Language Learner.

1 Introduction

Motivated by the interest in probabilistic logic learning [De Raedt *et al.*, 2008; De Raedt and Kimmig, 2013], and statistical relational learning (SRL) [Getoor and Taskar, 2007], we revisit the classical rule learning problem, but focus on a probabilistic instead of the classical deterministic setting pursued in inductive logic programming (ILP). Compared to inductive logic programming and traditional rule learners, we study an upgraded setting in which Prolog is replaced by a probabilistic Prolog, called ProbLog, and in which the rules, background theory and examples are all *probabilistic* rather than deterministic. The upgraded setting has the property that when all probabilities are set to 0 and 1, it corresponds to the standard ILP problem, as addressed by FOIL [Quinlan, 1990] and Progol [Muggleton, 1995].

In probabilistic rule learning, the task is to learn rules from examples that have both a probabilistic description and a probabilistic classification. In terms of ILP, this means that all atoms in the example description and target atom have a probability. The learned rules are used for predicting the probability of the target predicate (the output) given the probabilities in the description of the example (the inputs). This prediction task differs from standard rule-learning and ILP in that the hypotheses generated by ILP systems require the inputs to be deterministic and usually the prediction as well (although some rule-learners output the confidence of their prediction). In SRL systems, on the other hand, it is possible to provide probabilistic inputs and make a probabilistic prediction for a specific example. However, SRL systems do not learn rules or structure from probabilistic examples and they also do not upgrade the traditional rule learning or ILP setting, see Section 5 for more details.

The probabilistic rule learning setting is useful in any situation in which both the example descriptions and their classification are probabilistic (or uncertain). This arises naturally when example descriptions are produced through perception. For instance, in vision or autonomous agents, the example description can be the image description or the belief state, which is often produced using components that have been learned themselves to detect certain objects, relationships or measurements and which typically also indicate the reliability of the description. Similarly, when crawling and parsing the web, one obtains descriptions or parses of the involved texts that are uncertain themselves. Yet in all these situations it can be beneficial to learn rules that capture interrelationships between the predicates and that allow to predict in a reliable way particular target predicates. We shall illustrate this in the context of NELL, the Never-Ending Language Learner [Carlson *et al.*, 2010]. Probabilistic rule learning applies also naturally to probabilistic databases, which consist by definition of probabilistic facts. As two final examples, let us mention the work by [Chen *et al.*, 2008], who argue that probabilistic examples arise naturally when performing scientific experiments as the outcome of experiments may be uncertain, and a medical scenario in which doctors may describe all they know about a particular patient in terms of (subjective) probabilities. As such they might provide, in addition to some deterministic descriptions, statements about their belief in the outcome of an expensive test on a patient as well as their belief that the patient will survive the next five years.

We contribute an integrated approach to learning probabilistic relational rules from probabilistic examples and background knowledge. It is incorporated in the ProbFOIL$^+$ system and builds on ProbLog [De Raedt *et al.*, 2007], a simple probabilistic Prolog, and FOIL [Quinlan, 1990], and employs well-known principles and heuristics of rule learning [Lavrac *et al.*, 2012]. The key contributions of ProbFOIL$^+$ are 1) that it upgrades a traditional inductive logic programming system towards a probabilistic setting, in which the example descriptions, targets and learned rules are all probabilistic, 2) that the predicted probability for a target instance is the probability with which the instance is true (the degree of be-

[*] All authors contributed equally to this work.

[†] Now at Siemens AG, Otto-Hahn-Ring 6, GE-81739 Munich

[‡] Now at Sirris, A. Reyerslaan 80, BE-1030 Brussels

lief), 3) that it tightly integrates the rule and weight learning in a one-step process, rather than learning these separately, and 4) that it efficiently computes the probability of candidate clauses, without resorting to an expensive optimization approach, which is enabled by combining the traditional sequential covering approach with a local optimization step to compute the weights. ProbFOIL$^+$ differs significantly from existing SRL approaches as we shall argue in Section 5.

This paper is organized as follows. We define the probabilistic rule learning problem in Section 2. Section 3 describes ProbFOIL^{+1}, our algorithm for learning probabilistic rules from probabilistic data. In Section 4 our approach is evaluated with a case study on NELL, the Never-Ending Language Learner [Carlson *et al.*, 2010]. An overview of related work can be found in Section 5, and conclusions in Section 6.

2 Background and Problem Specification

We first introduce basic concepts of logic programming (Prolog). An *atom* $p(t_1, \cdots, t_n)$ consists of a *predicate* p/n of arity n and *terms* $t_1, \cdots, t_n$. A term is a (lowercase) *constant*, an (uppercase) *variable*, or a *functor*. A *literal* is an atom or its negation. A *clause* is a conjunction of literals. A *definite clause* contains exactly one non-negated atom, and is denoted $h \leftarrow b_1, \cdots, b_n$. The head h and the body $b_1, \cdots, b_n$ consists of atoms h, b_i. A *substitution* $\theta = \{V_1/t_1, \cdots, V_m/t_m\}$ maps variables V_i to terms t_i. Applying θ to an atom a, denoted $a\theta$, replaces all occurrences of V_i in a by t_i. A *grounding substitution* removes all logical variables.

ProbLog is a probabilistic Prolog that allows one to work with probabilistic facts and background knowledge [De Raedt *et al.*, 2007]. A ProbLog program consists of a set of definite clauses D and a set of probabilistic facts $p_i :: c_i$, which are facts c_i labeled with the probability p_i that their ground instances $c_i\theta$ are true. Given a finite set of grounding substitutions $\{\theta_{j1}, \ldots \theta_{ji_j}\}$ for each probabilistic fact $p_j :: c_j$, consider the set of ground facts $L_T = \{c_1\theta_{11}, \ldots, c_1\theta_{1i_1}, \cdots, c_n\theta_{n1}, \ldots, c_n\theta_{ni_n}\}$. A ProbLog program $T = \{p_1 :: c_1, ..., p_n :: c_n\}$ defines a probability distribution over subsets of facts $L \subseteq L_T$ as

$$P(L \mid T) = \prod_{c_i\theta_j \in L} p_i \prod_{c_i\theta_j \in L_T \setminus L} (1 - p_i).$$

In combination with the clauses D, ProbLog then defines the *success probability* of a query q to be

$$P_s(T \models q) = \sum_{\substack{L \subseteq L_T \\ L \cup D \models q}} P(L \mid T).$$

In other words, the probability of q is the probability that q is *entailed* using the background knowledge together with a random set of ground probabilistic facts.

Definition 1 *(Probabilistic Rule Learning)* **Given:**

1. *a set of examples E, consisting of pairs (x_i, p_i), where x_i is a ground fact for the unknown target predicate t and p_i is a target probability;*
2. *a background theory B containing information about the examples in the form of a ProbLog program;*
3. *a loss function $loss(H, B, E)$, measuring the loss of a hypothesis (set of clauses) H w.r.t. B and E;*
4. *a space of possible clauses $\mathcal{L}_h$ specified using a declarative bias;*

Find: *A hypothesis $H \subseteq \mathcal{L}_h$ such that $H = \arg\min_H loss(H, B, E)$.*

This loss function aims at minimizing the standard error of the predictions, that is, $loss(H, B, E) = \sum_{(x_i,p_i)\in E} |P_s(B \cup H \models x_i) - p_i|$. It is a simple choice with the benefit that well-known concepts and notions from rule learning and classification directly carry over to the probabilistic case. Furthermore, this loss function is also used in Kearns and Schapire's probabilistic concept-learning framework [Kearns and Schapire, 1994], a generalization of Valiant's probably approximate correct learning framework to predicting the probability of examples rather than their class. The above definition generalizes that framework further by assuming that also the descriptions of the examples themselves are probabilistic, not just their classes.

Notice that this problem setting generalizes both traditional rule learning and ILP to a probabilistic setting. To see this, consider that in the original FOIL system, the background theory was specified as a set of ground facts $\{f_j\}$ and each example was a true or false fact for the target predicate x_i. In the present setting, the background theory consists of a set of probabilistic facts $\{p_j :: f_j\}$ (possibly together with a set of definite clauses) and the examples (x_i, p_i) are also labelled with their target probability. It should be clear that the original setting is obtained when all probabilities p_j and p_i are 0 or 1. This is in line with the theory of probabilistic logic learning [De Raedt, 2008]. The ILP setting obtained is that of learning from entailment because examples are facts that are entailed by the theory.

This problem setting was proposed initially in De Raedt and Thon [2010], where also a preliminary rule-learner called ProbFOIL was proposed and illustrated at work on two toy examples. Both the setting and the rule learner presented in the current paper extend the work of De Raedt and Thon [2010] as we account for probabilistic rules (of the form $x :: head \leftarrow body$) instead of the deterministic ones (with $x = 1$) in ProbFOIL, we determine the optimal weights x (cf. Section 3), and include an experimental evaluation using Bayesian networks and NELL, cf. Section 5.

3 ProbFOIL$^+$

Algorithm Our ProbFOIL$^+$ algorithm for learning probabilistic clauses is shown as Algorithm 1. It directly generalizes the mFOIL rule learner, and closely resembles the vanilla rule-learning algorithm (cf. Mitchell [1997]) in that it follows the typical sequential covering approach. The outer loop of the algorithm starts from an empty set of clauses and repeatedly adds clauses to the hypothesis until no more improvement is observed with respect to a *global scoring function*. The clause to be added is obtained by the function LEARN-

[1] ProbFOIL$^+$ and the datasets used in this paper in ProbFOIL$^+$ format can be downloaded from https://dtai.cs.kuleuven.be/software/probfoil/.

RULE, which greedily searches for the clause that maximizes a *local scoring function*, using the refinement operator ρ.

Algorithm 1 The ProbFOIL$^+$ learning algorithm.

```
 1: function PROBFOIL$^+$($target$)
 2:   $H := \emptyset$
 3:   while true do
 4:     $clause$ := LEARNRULE($H$, $target$)
 5:     if GSCORE($H$) < GSCORE($H \cup \{clause\}$) then
 6:       $H := H \cup \{clause\}$
 7:     else return $H$
 8: function LEARNRULE($H$, $target$)
 9:   $candidates := \{x :: target \leftarrow true\}$
10:   $best := (x :: target \leftarrow true)$
11:   while $candidates \neq \emptyset$ do
12:     $next\_cand := \emptyset$
13:     for all $x :: target \leftarrow body \in candidates$ do
14:       for all $refinement \in \rho(target \leftarrow body)$ do
15:         if not REJECT($H$, $best$, $x :: target \leftarrow body$) then
16:           $next\_cand := next\_cand \cup \{x :: target \leftarrow body \wedge$
17:                                          $refinement\}$
18:           if LSCORE ($H$, $x :: target \leftarrow body \wedge refinement$) >
19:                                          LSCORE($H$, $best$) then
20:             $best := (x :: target \leftarrow body \wedge refinement)$
21:     $candidates := next\_cand$
22:   return $best$
```

While ProbLog and Prolog assume that the rules are definite clauses, in ProbFOIL$^+$ we use probabilistic rules of the form $x :: target \leftarrow body$. This is short hand for a ProbLog program with the deterministic rule $target \leftarrow body \wedge prob(id)$ and probabilistic fact $x :: prob(id)$, where id is an identifier that refers to this particular rule. Notice that all facts for such rules are independent of one another, and also that the probability x will have to be determined by the rule learning algorithm. Each call to LSCORE returns the best score that can be achieved for any value of x, and when returning the best found rule in line 22, the value of x is fixed to the probability that yields the highest local score.

Scoring Functions ProbFOIL$^+$ uses (upgraded versions) of standard scoring functions for rule learning, though others can easily be adapted as well. As the *global scoring function*, which determines the stopping criterion of the outer loop, we use $accuracy_H = \frac{TP_H + TN_H}{M}$ where M is the size of the dataset. The *local scoring function* is based on the m-estimate, a variant of precision that is more robust against noise in the training data, which is defined as $m\text{-}estimate_H = \frac{TP_H + m\frac{P}{N+P}}{TP_H + FP_H + m}$, where m is a parameter of the algorithm, and P and N indicates the number of positive and negative examples in the dataset, respectively.

Both these metrics are based on the number of examples correctly classified as positive (true positives, TP) and the number of examples incorrectly classified as positive (false positives, FP), which form the basis of the contingency tables for classification, and which we now upgrade for use in a probabilistic setting. While in a deterministic setting, each example e_i has a 1/0 target classification, this now becomes a probability value p_i. This means that every example contributes p_i to the positive part of the dataset and $1 - p_i$ to the negative part of the dataset, which generalizes the deterministic setting with $p_i = 1$ for positive and $p_i = 0$ for negative examples. In general, we define the positive and negative parts of the dataset as $P = \sum_{i=0}^{M} p_i$ and $N = \sum_{i=0}^{M}(1 - p_i) = M - P$.

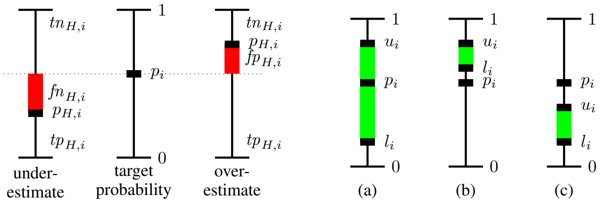

Figure 1: True/false and pos./neg. parts of a single example (left). Values for l_i, u_i and p_i where (a) it is still possible to perfectly predict p_i with the right value for x, or where p_i will always be (b) overestimated or (c) underestimated (right).

The same approach generalizes the predictions of a model to the probabilistic setting where a hypothesis H will predict a value $p_{H,i} \in [0, 1]$ for example e_i instead of 0 or 1. In this way we can define a probabilistic version of the true positive and false positive rates of the predictive model as $TP_H = \sum_{i=0}^{M} tp_{H,i}$, where $tp_{H,i} = \min(p_i, p_{H,i})$ and $FP_H = \sum_{i=0}^{M} fp_{H,i}$, where $fp_{H,i} = \max(0, p_{H,i} - p_i)$. For completeness we note that $TN_H = N - FP_H$ and $FN_H = P - TP_H$, as was the case in the deterministic setting.

Figure 1 illustrates these concepts. If a hypothesis H overestimates the target value of e_i, that is, $p_{H,i} > p_i$ then the true positive part tp_i will be maximal, that is, equal to p_i. The remaining part, $p_{H,i} - p_i$, is part of the false positives. If H underestimates the target value of e_i then the true positive part is only $p_{H,i}$ and the remaining part, $p_i - p_{H,i}$ contributes to the false negative part of the prediction.

Calculating x Algorithm 1 builds a set of clauses incrementally. Given a set of clauses H, it will search for the clause $c(x) = (x :: c)$ that maximizes the local scoring function, where $x \in [0, 1]$ is a multiplier indicating the probability that the body of c entails its head. The local score of the clause c is obtained by selecting the best possible value for x, that is, we want to find $\arg\max_x M(x)$ with

$$M(x) = \frac{TP_{H \cup c(x)} + m\frac{P}{N+P}}{TP_{H \cup c(x)} + FP_{H \cup c(x)} + m}$$

Next, we describe how to efficiently compute this value.

To find this optimal value, we need to be able to express the contingency table of $H \cup c(x)$ in function of x. As before, we use p_i to indicate the target value of example e_i.

We see that $p_{H \cup c(x),i}$ is a monotone function in x, that is, for each example e_i and each value of x, $p_{H \cup c(x),i} \geq p_{H,i}$ and for each x_1 and x_2, such that $x_1 \leq x_2$, it holds that $p_{H \cup c(x_1),i} \leq p_{H \cup c(x_2),i}$. The minimum and maximum prediction of $H \cup c(x)$ for the example e_i is thus

$$l_i = p_{H \cup c(0),i} = p_{H,i} \qquad u_i = p_{H \cup c(1),i}.$$

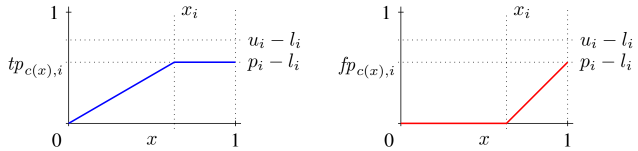

Figure 2: True and false positive rate for a single example $e_i \in E_3$ where $l_i < p_i < u_i$.

Note that u_i is the prediction that would be made by the original ProbFOIL algorithm [De Raedt and Thon, 2010], which learns deterministic rules.

For each example e_i, we can decompose $tp_{H \cup c(x),i}$ and $fp_{H \cup c(x),i}$ in

$$tp_{H \cup c(x)} = tp_{H,i} + tp_{c(x),i} \quad fp_{H \cup c(x),i} = fp_H + fp_{c(x),i},$$

where $tp_{c(x),i}$ and $fp_{c(x),i}$ indicate the additional contribution of clause $c(x)$ to the true and false positive rates.

Second, we divide the examples into three categories, as depicted in Figure 1:

E_1 : $p_i \leq l_i$, i.e., the clauses *overestimate* the target value for this example, irrespective of the value of x. For such an example $tp_{c(x),i} = 0$ and $fp_{c(x),i} = x(u_i - l_i)$.

E_2 : $p_i \geq u_i$, i.e., the clauses *underestimate* the target value for this example, irrespective of the value of x. For such an example $tp_{c(x),i} = x(u_i - l_i)$ and $fp_{c(x),i} = 0$.

E_3 : $l_i < p_i < u_i$, i.e., there exists a value of x for which the clause predicts the target value for this example *perfectly*. We call this value x_i and it can be computed as

$$x_i = \frac{p_i - l_i}{u_i - l_i}.$$

Figure 2 shows the values of $tp_{c(x),i}$ and $fp_{c(x),i}$ in function of x. The formulae for these functions are

$$tp_{c(x),i} = \begin{cases} x(u_i - l_i) & \text{if } x \leq x_i, \\ p_i - l_i & \text{if } x > x_i \end{cases}$$

and

$$fp_{c(x),i} = \begin{cases} 0 & \text{if } x \leq x_i, \\ x(u_i - l_i) - (p_i - l_i) & \text{if } x > x_i \end{cases}.$$

We can now determine the contribution to $TP_{c(x)}$ and $FP_{c(x)}$ of the examples in each of these categories. For the examples in E_1, the contributions to $TP_{c(x)}$ and $FP_{c(x)}$ are

$$TP_1(x) = 0 \text{ and } FP_1(x) = x \sum_i^{E_1} (u_i - l_i) = xU_1.$$

For the examples in E_2, the contribution to $TP_{c(x)}$ and $FP_{c(x)}$ are

$$TP_2(x) = x \sum_i^{E_2} (u_i - l_i) = xU_2 \text{ and } FP_2(x) = 0.$$

For the examples in E_3, the contributions to $TP_{c(x)}$ and $FP_{c(x)}$ are

$$TP_3(x) = x \sum_{i:x \leq x_i}^{E_3} (u_i - l_i) + \sum_{i:x > x_i}^{E_3} (p_i - l_i) = xU_3^{\leq x_i} + P_3^{> x_i},$$

$$FP_3(x) = x \sum_{i:x > x_i}^{E_3} (u_i - l_i) - \sum_{i:x > x_i}^{E_3} (p_i - l_i) = xU_3^{> x_i} - P_3^{> x_i}.$$

By using the fact that $TP_{H \cup c(x)} = TP_H + TP_1(x) + TP_2(x) + TP_3(x)$ and $FP_{H \cup c(x)} = FP_H + FP_1(x) + FP_2(x) + FP_3(x)$ and by reordering terms we can reformulate the definition of the m-estimate as

$$\begin{aligned} M(x) &= \frac{TP_{H \cup c(x)} + m\frac{P}{N+P}}{TP_{H \cup c(x)} + FP_{H \cup c(x)} + m} \\ &= \frac{(U_2 + U_3^{\leq x_i})x + TP_H + P_3^{> x_i} + m\frac{P}{N+P}}{(U_1 + U_2 + U_3)x + TP_H + FP_H + m}. \end{aligned} \quad (1)$$

In the last step we replaced $FP_3(x) + TP_3(x) = x \sum_i^{E_3} (u_i - l_i) = xU_3$.

By observing that $U_3^{\leq x_i}$ and $P_3^{> x_i}$ are constant on the interval between two consecutive values of x_i, we see that this function is a piecewise non-linear function where each segment is of the form

$$\frac{Ax + B}{Cx + D}$$

where A, B, C and D are constants. The derivative of such a function is

$$\frac{dM(x)}{dx} = \frac{AD - BC}{(CX + D)^2},$$

which is non-zero everywhere or zero everywhere. This means that the maximum of $M(x)$ will occur at one of the endpoints of the segments, that is, in one of the points x_i. By incrementally computing the values of $U_3^{\leq x_i} = \sum_{i:x \leq x_i}^{E_3} (p_i - l_i)$ and $P_3^{> x_i} = \sum_{i:x > x_i}^{E_3} (p_i - l_i)$ in Equation 1 for the x_i in increasing order, we can efficiently find the value of x that maximizes the local scoring function. Moreover, by computing one probability $u_i = p_{H \cup c(1),i}$ for each example e_i, we can obtain all probabilities for different x, as $p_{H \cup c(x),i} = l_i + x(u_i - l_i)$.

Significance In order to avoid learning large hypotheses with many clauses that only have limited contributions, we use a significance test. This test was also used in the mFOIL algorithm [Džeroski, 1993]. It is a variant of the likelihood ratio statistic and is defined as

$$\begin{aligned} LhR(H, c) = 2(TP_{H,c} + FP_{H,c}) \\ \left(prec_{H,c} \log \frac{prec_{H,c}}{prec_{true}} + (1 - prec_{H,c}) \log \frac{1 - prec_{H,c}}{1 - prec_{true}} \right), \end{aligned}$$

where

$$TP_{H,c} = TP_{H \cup c} - TP_H, \quad FP_{H,c} = FP_{H \cup c} - FP_H$$

$$prec_{H,c} = \frac{TP_{H,c}}{TP_{H,c} + FP_{H,c}}, \quad prec_{true} = \frac{P}{P + N}.$$

This statistic is distributed according to χ^2 with one degree of freedom. Note that we use a relative likelihood, which is based on the additional prediction made by adding clause c to hypothesis H. As a result, later clauses will automatically achieve a lower likelihood.

Local stopping criteria When we analyze the algorithm above, we notice that in the outer loop, the number of positive predictions increases. This means that the values in the first row of the contingency table can only increase (and the values in the second row will decrease). More formally:

Property 1 *For all hypotheses H_1, H_2: $H_1 \subset H_2 \rightarrow TP_{H_1} \leq TP_{H_2}$ and $FP_{H_1} \leq FP_{H_2}$.*

Additionally, in the inner loop, we start from the most general clause (i.e., the one that always predicts 1), and we add literals to reduce the coverage of negative examples. As a result, the positive predictions will decrease.

Property 2 *For all hypotheses H and clauses $h \leftarrow l_1, ..., l_n$ and literals l: $TP_{H\cup\{h\leftarrow l_1,...,l_n\}} \leq TP_{H\cup\{h\leftarrow l_1,...,l_n,l\}}$ and $FP_{H\cup\{h\leftarrow l_1,...,l_n\}} \leq FP_{H\cup\{h\leftarrow l_1,...,l_n,l\}}$*

We can use these properties to determine when a refinement can be rejected (line 15 of Algorithm 1). In order for a clause to be a viable candidate it has to have a refinement that 1) has a higher local score than the current best rule, 2) has a significance that is high enough (according to a preset threshold), and 3) has a better global score than the current rule set without the additional clause.

Implementation Details As usual in ILP, ProbFOIL$^+$ uses a declarative bias based on modes [Muggleton, 1995]. These specify syntactic restrictions on the clauses of interest and are used by the refinement operator during the search process.

The LEARNRULE function of the ProbFOIL$^+$ algorithm is based on mFOIL [Džeroski, 1993] and uses a beam search strategy in order to escape from local maxima. It uses relational path finding [Ong *et al.*, 2005; Richards and Mooney, 1992] to generate clauses by considering the connections between the variables in the example literals, a proven technique to direct the search in first-order rule learning.

ProbFOIL$^+$ computes the probabilities $p_{H,i}$ using the ProbLog2 system [Fierens *et al.*, 2014].[2] The ProbLog2 inference engine computes the probability of queries in four phases: it grounds out the relevant part of the probabilistic program, converts this to a CNF form, performs knowledge compilation into d-DNNF form and, finally, computes the probability from the obtained d-DNNF structure. This process is described in detail in [Fierens *et al.*, 2014].

Due to the specific combinations and structure of ProbFOIL$^+$'s queries, we can apply multiple optimizations[3]:
Incremental grounding While the standard ProbLog2 would perform grounding for each query, ProbFOIL$^+$ uses incremental grounding techniques and builds on the grounding from the previous iteration instead of starting from scratch. This is possible as the rules are constructed and evaluated one literal at-a-time.
Direct calculation of probabilities Because of the incremental nature of ProbFOIL$^+$'s evaluation, we can often directly compute probabilities without having to resort to (costly) knowledge compilation, for example when we add a literal whose grounding does not share facts with the grounding of the rest of the theory (for which we computed the probability in a previous iteration). This can also significantly reduce the size of the theories that need to be compiled.
Propositional data When propositional data is used, all examples have the same structural component. This means we can construct a d-DNNF for a single example and reuse it to evaluate all other examples.
Range-restricted rules Since in a number of cases, it is desired that the result in rules are *range-restricted*, i.e., that all variables appearing in the head of a clause also appear in its body, ProbFOIL$^+$ offers an option to output only range-restricted rules.

4 Experiments

We answer two questions experimentally.

Q1: How do ProbFOIL and ProbFOIL$^+$ compare to standard regression learners in the propositional case? This question is motivated by the observation that – in the propositional case – the task can be viewed as that of predicting the probability of the target example from a set of probabilistic attributes, which can be solved by applying standard regression tasks. Of course, one then obtains regression models, which do not take the form of a set of logical rules that are easy to interpret. While regression can in principle be applied to the propositional case, it is hard to see which regression systems would apply to the relational case. The reason is that essentially *all* predicates are probabilistic (and hence, numeric), a situation that is – to the best of the authors' knowledge – unprecedented in relational learning. Standard relational regression algorithms are able to predict numeric values starting from a relational description, a set of true and false ground facts. The goal of this experiment is not to suggest that probabilistic rule learning can contribute to regression, it is rather that regression provides a reasonable baseline that allows to evaluate the performance of probabilistic rule-learning.

Dataset We will generate data from Bayesian networks, both for dependent and independent attributes, and partial and full observability. The *target variable* will be the variable one wants to predict, this will always be a node that does not have children in the network. The evidence or descriptor variables will be a subset of the other variables.We use BNGenerator[4] to randomly generate a Bayesian network structure. The conditional probability tables (CPT) and marginal distributions are left unspecified. The generated network has 45 nodes, 70 edges, a maximal degree of 6 and an induced width of 5.

[2] http://dtai.cs.kuleuven.be/problog/

[3] The remainder of the section can be skipped by the reader less familiar with probabilistic programming

[4] http://www.pmr.poli.usp.br/ltd/Software/BNGenerator/

Table 1: Mean absolute error on the Bayesian network with CPTs $\sim \mathrm{Beta}(\alpha, \beta)$, averaged over all target attributes.

	independent			dependent full observability			dependent partial observability		
α, β	1.0000	0.0100	0.001	1.0000	0.0100	0.001	1.0000	0.0100	0.001
ZeroR	0.023	0.085	0.093	0.023	0.085	0.093	0.023	0.085	0.093
LinearRegression	4.4×10^{-3}	0.022	0.023	2.6×10^{-3}	0.021	0.020	4.8×10^{-3}	0.026	0.032
MultilayerPerceptron	9×10^{-4}	7.2×10^{-3}	6.7×10^{-3}	4×10^{-4}	5.7×10^{-3}	3.9×10^{-3}	1.2×10^{-3}	9.7×10^{-3}	0.020
M5P	1.5×10^{-3}	8.8×10^{-3}	8.7×10^{-3}	7×10^{-4}	6.5×10^{-3}	5.4×10^{-3}	1.6×10^{-3}	0.022	0.027
M5P -R -M 4.0	5.8×10^{-3}	0.025	0.028	5.2×10^{-3}	0.023	0.025	6.2×10^{-3}	0.033	0.040
SMOreg	4.4×10^{-3}	0.022	0.023	2.6×10^{-3}	0.021	0.020	4.5×10^{-3}	0.022	0.029
ProbFOIL	0.077	5.2×10^{-3}	9.4×10^{-8}	0.015	1.9×10^{-3}	9.4×10^{-8}	0.020	0.012	0.015
ProbFOIL$^+$	2.3×10^{-3}	2.9×10^{-4}	9.4×10^{-8}	3.9×10^{-3}	5.3×10^{-4}	2.8×10^{-7}	9.5×10^{-3}	0.011	0.013

Subsequently, different instances of the Bayesian network are generated by sampling its CPTs from a beta distribution $\mathrm{Beta}(\alpha, \beta)$. Lower values for α and β make the network more "deterministic" and less "probabilistic". To generate training and test examples for a single network instance, we uniformly sample marginal probabilities for the root nodes. These values, together with the inferred probability of the target, make up a single example. Each combination of target attribute and beta distribution is a different learning problem. For each of these, we trained ProbFOIL, ProbFOIL$^+$ and standard regression learners from the Weka suite on 500 training examples. The learned models are evaluated on 500 test examples using the mean absolute error, which is 1 minus the accuracy in the rule learning setting. We consider also three settings, whose results are shown in Table 1:

1. *Independent Attributes.* In this simplest setting, the observed attributes are all root nodes of the Bayesian network, i.e., a node with no parent nodes in the graph.
2. *Dependent Attributes, Full Observability.* In this setting, the observed nodes are no longer the root nodes. Consequently, their probabilities are not independent anymore. There is, however, an observed node on every path from a root node to a target node, allowing for full observability and the possibility of rediscovering the model the data was drawn from.
3. *Dependent Attributes, Partial Observability.* By dropping full observability, we can no longer learn the perfect model.

Answer to Q1: In almost all cases ProbFOIL$^+$ performs on par or outperforms the standard regression learners, which demonstrates its advantage for propositional probabilistic rule learning. Furthermore, in all cases, similar or better results are obtained by ProbFOIL$^+$ when compared to ProbFOIL, illustrating the added value of learning probabilistic rules with weights.

Q2: How does ProbFOIL$^+$ perform for relational probabilistic rule learning in the context of a probabilistic knowledge base? The task to extract information from unstructured or semi-structured documents has recently attracted an increased amount of attention in the context of Machine Reading. Here we focus on NELL[5], to which several rule learning approaches have already been applied, as discussed in Section 5.

Table 2: Number of facts per predicate (NELL sports dataset) for predicates used in the learned rules (Table 3).

athleteledsportsteam(athlete,team)	246
athleteplaysforteam(athlete,team)	808
athleteplaysinleague(athlete,league)	1197
athleteplayssport(athlete,sport)	1899
teamalsoknownas(team,team)	273
teamplaysagainstteam(team,team)	2848
teamplayssport(team,sport)	340
teamplaysinleague(team,league)	1229

Dataset In order to test probabilistic rule learning for NELL, we extracted the facts for all predicates related to the sports domain from iteration 850 of the NELL knowledge base[6]. A similar dataset was used in the context of meta-interpretive learning [Muggleton and Lin, 2013]. Our dataset contains 10567 facts. The number of facts per predicate (and their types) are listed in Table 2. Each fact has a probability value attached (e.g., $0.934 :: athleteplaysforteam(thurman\_thomas, buffalo\_bills)$). Part of the negative examples are the negative contribution of the positive examples ($1 - probability$). The additional negative examples are generated by taking random combinations of constants present in the dataset (while respecting the type information). To reduce the search space during rule learning, we impose the constraint that each literal can introduce at most one new variable as well as range-restrictedness.

To evaluate ProbFOIL$^+$ in the context of NELL, we learned rules for each binary predicate[7] with more than 500 facts. In order to have a reasonable number of examples per fold, we used 3-fold cross-validation. To create the folds, for each target predicate, the facts were randomly split into 3 parts. Each fold consists of all non-target predicates and a part of the target predicates. Due to space limitations, we only report the rules for three of the predicates that are learned on

[5] See http://rtw.ml.cmu.edu

[6] From http://rtw.ml.cmu.edu/rtw/resources. Iteration 850 was the last available iteration at the time of experimentation.

[7] Note that for the presented algorithm, the target predicates are not restricted to binary only. This just happens to be the case in the dataset we use.

Table 3: Learned relational rules for the different predicates (fold 1).

0.9375::athleteplaysforteam(A,B)	←	athleteledsportsteam(A,B).
0.9675::athleteplaysforteam(A,B)	←	athleteledsportsteam(A,V1), teamplaysagainstteam(B,V1).
0.9375::athleteplaysforteam(A,B)	←	athleteplayssport(A,V1), teamplayssport(B,V1).
0.5109::athleteplaysforteam(A,B)	←	athleteplaysinleague(A,V1), teamplaysinleague(B,V1).
0.9070::athleteplayssport(A,B)	←	athleteledsportsteam(A,V2), teamalsoknownas(V2,V1), teamplayssport(V1,B), teamplayssport(V2,B).
0.9070::athleteplayssport(A,B)	←	athleteplaysforteam(A,V2), teamalsoknownas(V2,V1), teamplayssport(V1,B), teamplayssport(V2,B),teamalsoknownas(V1,V2).
0.9070::athleteplayssport(A,B)	←	athleteplaysforteam(A,V1), teamplayssport(V1,B).
0.9286::athleteplaysinleague(A,B)	←	athleteledsportsteam(A,V1), teamplaysinleague(V1,B).
0.7868::athleteplaysinleague(A,B)	←	athleteplaysforteam(A,V2), teamalsoknownas(V2,V1), teamplaysinleague(V1,B).
0.9384::athleteplaysinleague(A,B)	←	athleteplayssport(A,V2), athleteplayssport(V1,V2), teamplaysinleague(V1,B).
0.9024::athleteplaysinleague(A,B)	←	athleteplaysforteam(A,V1), teamplaysinleague(V1,B).

Table 4: Precision for different experimental setups and parameters (*A*: $m = 1, p = 0.99$, *B*: $m = 1000, p = 0.90$).

Setting	**athleteplaysforteam**		**athleteplayssport**		**teamplaysinleague**		**athleteplaysinleague**		**teamplaysagainstteam**	
train/test/rule	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**
1: det/det/det	74.00	69.36	94.14	93.47	96.29	82.15	80.95	74.14	73.40	73.86
2: det/prob/det	73.51	69.57	97.53	94.85	96.70	87.83	90.83	77.73	73.70	73.35
3: det/prob/prob	74.67	69.82	95.86	94.74	96.35	82.57	82.26	75.29	73.84	74.34
4: det/prob/prob	77.25	73.87	96.53	96.04	98.00	90.59	84.91	79.36	77.26	77.83
5: det/prob/prob	74.76	69.97	95.85	94.69	96.44	82.51	81.99	75.07	73.90	74.16
6: prob/prob/det	75.83	73.11	93.40	93.76	94.44	93.67	79.41	79.42	80.87	80.60
7: prob/prob/prob	78.31	73.72	95.62	95.10	98.84	91.86	96.94	79.49	85.78	81.81

the first fold. Similar rules were obtained on the other predicates and folds.

Experimental set-up. Our experimental set-up is motivated as follows. As baselines for probabilistic rule learning, we chose several special cases of ProbFOIL$^+$, possibly combined with other techniques. Each of these special cases closely corresponds to an approach that exists in the literature, and hence, can be used as a baseline. This provides a more controlled experimental setting than a comparison with other ILP or SRL systems. Furthermore, a comparison with other ILP or SRL systems would be problematic as we are not aware of other systems that can cope with both probabilistic descriptions and probabilistic classifications. In addition, when considering, for instance, Markov Logic, it would be unclear how to turn the probabilities of atoms in the example descriptions into weights for use by Markov Logic.

Setting 1 is the fully deterministic case. If ProbFOIL$^+$ uses fully deterministic examples, this directly corresponds to mFOIL and thus is representative of the pure ILP approach. To this end, as usual in NELL, we interpret each example with a probability higher than 0.75 as a positive one, and each example with a lower probability as a negative one. Setting 2 uses the same learning algorithm and data as in Setting 1 but uses the learned rules with probabilistic inputs to produce a probabilistic classification. Setting 3 is a variant of Setting 1 in which we first learn the rules in a purely deterministic way (as in Settings 1 and 2), and then assign a weight to each learned rule. The weight is the precision of the rule estimated using the probabilities of the target in the training set. This closely corresponds to what some rule and decision tree learners do, namely estimating the probability of class membership in the conclusion part of the rule or in the leaves of the decision tree. These weighted rules can then be used for probabilistic class prediction. Note that in this setting only the class probability of the example is taken into account, not the probabilities of the descriptors. Setting 4 extends the previous setting in that it also takes into account the probabilistic example descriptions for computing the weights. While in Setting 3, when the body of a fully deterministic rule is true, one would always predict a probability of 1, in Setting 4 the predicted probability is the probability with which the body of the rule is true in the example. In Setting 5 we first learn deterministic rules and then train the weights with LFE using a least-squares approach. LFE is a learning technique for parameter estimation that naturally works with probabilistic inputs and probabilistic outputs, cf. [Gutmann *et al.*, 2008]. This setting closely mimics two step approaches such as those of Schoenmackers *et al.* [2010], N-FOIL [Lao *et al.*, 2011], and Raghavan *et al.* [2012] in that one first learns deterministic rules and in a second step learns their weights or probabilities, see also Section 5 for a discussion of these appraoches. Setting 6 uses probabilistic examples in both training and test set, but uses ProbFOIL, the deterministic version of ProbFOIL$^+$. Setting 7 then corresponds to the full ProbFOIL$^+$ setting.

Similar to previous related work (e.g., Carlson *et al.* [2010], Schoenmackers *et al.* [2010], Raghavan and Mooney [2013]) we used precision as our primary evaluation measure. It measures the fraction of the probabilistic inferences that are deemed correct. Measuring the true recall is impossible in this context, since it would require *all* correct facts for a given target predicate. For example, it is possible that correct facts are inferred using the obtained rules, which are not (yet) present in the knowledge base, and consequently are not reflected in the recall score.

For all predicates, the m-estimate's m value was set to 1 and the beam width to 5. The value of p for rule significance was set to 0.99. Furthermore, to avoid a bias towards the majority class, the examples are balanced, i.e., a part of the negative examples is removed.

Discussion As is clear from Table 3, ProbFOIL$^+$ learns interpretable rules. Some of the rules are less meaningful than others, which can be explained by the small number of constants (in this case representing entities related to athletes and teams) in the dataset. In all cases, ProbFOIL$^+$ performs on par or outperforms the baselines that use deterministic training data, and ProbFOIL. In order to avoid overfitting because of over-specific rules, we also tested all settings with a high m-value (1000), and a rule significance p of 0.9 (parameter setting B). This also limits the capability of the algorithm to fit to small variations that are actually improving the predictive power. However, one can observe that ProbFOIL$^+$ is still able to perform similarly or better than the other settings. With these settings, ProbFOIL$^+$ learns more deterministic rules. This can also be seen from results obtained with ProbFOIL (Setting 6), which are now more similar to the ones obtained with ProbFOIL$^+$. Furthermore, the obtained rule sets achieve similar results on training and test set, indicating the generalizability of the learned rules.

The evaluation for the machine reading setting is limited by the available data, which should be taken into account when interpreting these results. First of all, the distribution of the probabilities in the NELL dataset is very skewed. Moreover, the dataset also contains a number of predicates for which only a small number of facts are available in knowledge base. Finally, the confidence scores that are currently attached to the facts in NELL are a combination of the probability output by the learning algorithm and a manual evaluation. Even under these circumstances, ProbFOIL$^+$ performs better than a purely deterministic or two-step approach.

Answer to Q2: ProbFOIL$^+$ obtains promising results for relational probabilistic rule learning. Its use can be valuable for expanding a probabilistic knowledge base, as illustrated in the context of NELL.

5 Related Work

There is a large body of related research, much of which originates from the machine reading domain or from statistical relational learning. To the best of the authors' knowledge, none of these possess the combination of the four features listed at the end of the introduction.

First, there are several works that aim at learning inference rules from automatically extracted data and using the learned rules to expand the knowledge base in NELL. Most of these approaches (like N-FOIL [Lao *et al.*, 2011], the approach to learning Bayesian Logic Programs (BLPs) of Raghavan *et al.* [2012], and Schoenmackers *et al.* [2010]) all proceed in two steps (and hence do not satisfy feature 3). In the first step, a deterministic rule-learner is applied to a deterministic setting, and the weights are then determined in a second step using a variety of techniques, while we jointly optimize the rules and the parameters. The deterministic setting was also a baseline chosen in our experiment on the NELL data.

Secondly, another major difference lies in the underlying probabilistic logical framework that ranges from Markov Logic [Schoenmackers *et al.*, 2010] to BLPs [Raghavan *et al.*, 2012] and variations of stochastic logic programs (SLPs) [Lao *et al.*, 2011; Wang *et al.*, 2014; Chen *et al.*, 2008]). Markov Logic and BLPs are based on knowledge based construction and hence, correspond to graphical models, which sets it apart from approaches such as ProbLog based on logical deduction. Furthermore, the semantics of the approaches based on stochastic logic programs is quite different in that in Halpern's [1990] terminology, it is more a type 1 than type 2 probabilistic logic. Type 1 logics are similar to grammars, they determine the probability with which a sentence (or atom) would be sampled from the model, rather than a degree of belief in the truth-value of that sentence in the world (thus these SLP approaches do not satisfy 2). Another difference with [Chen *et al.*, 2008] is that they use an abductive rather than an inductive approach.

Thirdly, some approaches learn both the global structure and parameters of SRL models, and even do order the search using a form of θ-subsumption. However, none of these approaches directly upgrades the traditional ILP rule-learning setting. Instead they learn a full SRL model typically from (partial) interpretations instead of from entailment, that is, the examples are sets of ground facts rather than specific facts with an associated target probability (and hence do not satisfy 1). The techniques (and the scoring functions) are quite different for this case. Typically, a mixture of EM and a search for possible rules is used (e.g. Bellodi and Riguzzi [2012], Sorower *et al.* [2011]) (which does not satisfy 4).

Finally, a number of other extensions of FOIL exists. nFOIL [Landwehr *et al.*, 2007] integrates FOIL with the Naïve Bayes learning scheme, such that Naïve Bayes is used to guide the search. kFOIL [Landwehr *et al.*, 2010] is a propositionalization technique that uses a combination of FOIL's rule-learning algorithm and kernel methods to derive a set of features from a relational representation. To this end, FOIL searches relevant clauses that can be used as features in kernel methods. These approaches do not satisfy 1 and 3.

6 Conclusion

We have introduced a novel setting for probabilistic rule learning, in which probabilistic rules are learned from probabilistic examples. The ProbFOIL$^+$ algorithm we developed solves this problem by combining the principles of the rule learner FOIL with the probabilistic Prolog called ProbLog. The result is a natural probabilistic extension of ILP and rule learning. We evaluated the approach against regression learners, and showed results on both propositional and relational probabilistic rule learning. Furthermore, we explored its use for knowledge base expansion in the context of NELL.

Acknowledgements

The authors would like to thank Jesse Davis for his invaluable input, the reviewers for their useful suggestions, and the Research Foundation - Flanders (FWO) for its financial support.

References

[Bellodi and Riguzzi, 2012] Elena Bellodi and Fabrizio Riguzzi. Learning the structure of probabilistic logic programs. In *ILP*, volume 7207 of *LNCS*, pages 61–75. Springer, 2012.

[Carlson *et al.*, 2010] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

[Chen *et al.*, 2008] Jianzhong Chen, Stephen Muggleton, and José Santos. Learning probabilistic logic models from probabilistic examples. *Machine learning*, 73(1):55–85, 2008.

[De Raedt and Kimmig, 2013] Luc De Raedt and Angelika Kimmig. Probabilistic programming concepts. *CoRR*, abs/1312.4328, 2013.

[De Raedt and Thon, 2010] Luc De Raedt and Ingo Thon. Probabilistic rule learning. In *ILP*, volume 6489 of *LNCS*, pages 47–58, 2010.

[De Raedt *et al.*, 2007] L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic Prolog and its application in link discovery. In M. Veloso, editor, *IJCAI*, pages 2462–2467, 2007.

[De Raedt *et al.*, 2008] L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors. *Probabilistic Inductive Logic Programming — Theory and Applications*, volume 4911 of *LNAI*. Springer, 2008.

[De Raedt, 2008] L. De Raedt. *Logical and Relational Learning*. Springer, 2008.

[Džeroski, 1993] S. Džeroski. Handling imperfect data in inductive logic programming. In *SCAI*, pages 111–125, 1993.

[Fierens *et al.*, 2014] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *TPLP*, 2014.

[Getoor and Taskar, 2007] L. Getoor and B. Taskar, editors. *An Introduction to Statistical Relational Learning*. MIT Press, 2007.

[Gutmann *et al.*, 2008] B. Gutmann, A. Kimmig, K. Kersting, and L. De Raedt. Parameter learning in probabilistic databases: A least squares approach. In *ECML-PKDD*, pages 473–488. Springer, 2008.

[Halpern, 1990] J. Halpern. An analysis of first-order logics of probability. *AIJ*, 46(3):311–350, 1990.

[Kearns and Schapire, 1994] Michael J. Kearns and Robert E. Schapire. Efficient distribution-free learning of probabilistic concepts. *Computer and System Sciences*, 48(3):464–497, 1994.

[Landwehr *et al.*, 2007] Niels Landwehr, Kristian Kersting, and Luc De Raedt. Integrating Naïve Bayes and FOIL. *JMLR*, 8:481–507, May 2007.

[Landwehr *et al.*, 2010] Niels Landwehr, Andrea Passerini, Luc De Raedt, and Paolo Frasconi. Fast learning of relational kernels. *Machine Learning*, 78(3):305–342, 2010.

[Lao *et al.*, 2011] Ni Lao, Tom Mitchell, and William W. Cohen. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, pages 529–539, USA, 2011. ACL.

[Lavrac *et al.*, 2012] Nada Lavrac, Johannes Furnkranz, and Dragan Gamberger. *Foundations of rule learning*. Springer Berlin Heidelberg, 2012.

[Mitchell, 1997] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Muggleton and Lin, 2013] Stephen H. Muggleton and Dianhuan Lin. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. In *IJCAI*, 2013.

[Muggleton, 1995] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13(3-4):245–286, 1995.

[Ong *et al.*, 2005] Irene M. Ong, Inẽs de Castro Dutra, David Page, and Vítor Santos Costa. Mode directed path finding. In *ECML*, volume 3720 of *LNCS*, pages 673–681. Springer, 2005.

[Quinlan, 1990] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Raghavan and Mooney, 2013] Sindhu Raghavan and Raymond J. Mooney. Online inference-rule learning from natural-language extractions. In *StaRAI*, 2013.

[Raghavan *et al.*, 2012] Sindhu Raghavan, Raymond J. Mooney, and Hyeonseo Ku. Learning to “read between the lines” using Bayesian logic programs. In *ACL*, pages 349–358, 2012.

[Richards and Mooney, 1992] Bradley L. Richards and Raymond J. Mooney. Learning relations by pathfinding. In *AAAI*, pages 50–55, July 1992.

[Schoenmackers *et al.*, 2010] Stefan Schoenmackers, Oren Etzioni, Daniel S. Weld, and Jesse Davis. Learning first-order horn clauses from web text. In *EMNLP*, pages 1088–1098, Stroudsburg, PA, USA, 2010.

[Sorower *et al.*, 2011] Mohammad S. Sorower, Janardhan R. Doppa, Walker Orr, Prasad Tadepalli, Thomas G Dietterich, and Xiaoli Z. Fern. Inverting grice’s maxims to learn rules from natural language extractions. In *NIPS*, pages 1053–1061, 2011.

[Wang *et al.*, 2014] William Y. Wang, Kathryn Mazaitis, and William W. Cohen. Structure learning via parameter learning. In *CIKM*, 2014.