The Gradient Semiring for Probabilistic Answer Set Programming and Its Application to Parameter Learning

Elisabetta Gentili[1⋆][0009−0006−6901−0540], Alice Bizzarri[1⋆][0000−0001−9259−7761], Damiano Azzolini[1][0000−0002−7133−2673], and Fabrizio Riguzzi[1][0000−0003−1654−9703]

University of Ferrara, Ferrara, Italy
{elisabetta.gentili1,alice.bizzarri,damiano.azzolini,fabrizio.riguzzi}@unife.it

Abstract. Semirings have been widely used in the last years to describe in an abstract way various tasks in Statistical Relational Artificial Intelligence. Here, we focus on the Probabilistic Answer Set Programming (PASP) formalism and propose an algebraic characterization of parameter learning, where the goal is to tune the probabilities of a program to model a set of examples. We introduce the gradient semiring for PASP and implement it on top of a state-of-the-art solver. A preliminary experimental evaluation shows the effectiveness of our approach.

Keywords: Probabilistic Answer Set Programming · Parameter Learning · Algebraic Model Counting.

1 Introduction

Statistical Relational AI (StarAI) [26] is a research area aimed at integrating probability with logical reasoning, combining the expressiveness of symbolic logic with the ability of probabilistic models to handle uncertainty. The languages used in StarAI include Markov Logic Networks [27], Probabilistic Logic Programming [14,28], and Probabilistic Answer Set Programming (PASP) [11], the latter being the focus of this work.

PASP extends traditional Answer Set Programming with probabilistic facts, allowing one to model scenarios where information is uncertain or incomplete. Here, we focus on the task of parameter learning, where the goal is to tune the parameters (i.e., the probabilities) of the probabilistic facts of a given PASP to maximize the probability of a set of examples. This problem is typically solved using Expectation-Maximization (EM) [3,7,16] or Gradient Descent [23].

Recently, semirings have been proposed as a general framework for various reasoning and learning tasks in StarAI [25,24]. In this paper, we propose an approach that extends the gradient semiring to solve the parameter learning task in PASP, and implement it on top of a state-of-the-art solver for, namely aspmc [17]. Preliminary experiments demonstrate the validity of our approach.

⋆ equal contribution

The paper is structured as follows: Section 2 provides background knowledge, Section 3 describes our approach, Section 4 reports the results of the experiments, Section 5 discusses related work, and Section 6 draws the final conclusions.

2 Background

Here we introduce the basic concepts of Probabilistic Answer Set Programming (PASP and we will use the same acronym to denote a program in this formalism) and Algebraic Model Counting.

2.1 Probabilistic Answer Set Programming

ProbLog *probabilistic facts* [14] are well-known syntactic constructs used to express uncertainty with a logic program. A probabilistic fact is of the form $\pi :: a$ where $\pi \in [0, 1]$ is the probability associated with the atom a. Without loss of generality, we consider only *ground* probabilistic facts. A probabilistic answer set program [11] is an answer set program (ASP) [9,20] extended with probabilistic facts. A *world* is an ASP obtained from the PASP by including or not each probabilistic fact (if n is the number of ground probabilistic facts, there are 2^n worlds), and the semantics assigns each world a set of answer sets. The *credal semantics* [10], which we consider in this paper, requires that each world has *at least one* answer set (also called stable model). The probability of a world w, $P(w)$, is computed as

$$P(w) = \prod_{a \in w} \pi \prod_{a \notin w} (1 - \pi)$$

since probabilistic facts are assumed to be independent. A *query* is a conjunction of ground literals (i.e., atoms or negated atoms, denoted by prepending *not*). The probability of a query q in a PASP is described by a lower $\underline{\mathrm{P}}(q)$ and an upper $\overline{\mathrm{P}}(q)$ probability computed as

$$\underline{\mathrm{P}}(q) = \sum_{w_i | \forall M \in AS(w_i),\ M \models q} P(w_i), \quad \overline{\mathrm{P}}(q) = \sum_{w_i | \exists M \in AS(w_i),\ M \models q} P(w_i)$$

where $AS(w_i)$ denotes the set of answer sets for the world w_i. That is, the lower probability is the sum of the probabilities of the world where the query is true in *every* answer set (i.e., it is a *cautious* consequence) while the upper probability is the sum of the probabilities of the world where the query is true in *at least one* answer set (i.e., it is a *brave* consequence).

For further clarification, consider the PASP presented in Example 1.

Example 1. This PASP encodes a simple graph reachability problem.

```
0.2::edge(1,2). 0.3::edge(2,4). 0.9::edge(1,3).
path(X,Y):- connected(X,Z), path(Z,Y).
path(X,Y):- connected(X,Y).
```

```
connected(X,Y):- edge(X,Y), not nconnected(X,Y).
nconnected(X,Y):- edge(X,Y), not connected(X,Y).
```

The first three lines define probabilistic facts, where each $edge/2$ fact is associated with the probability of existence of an edge between two nodes. The $path/2$ predicate defines a path recursively: a path exists between two nodes X and Y if there is a direct connection between them ($connected(X, Y)$), or there is an intermediate node Z such that both $connected(X, Z)$ and $path(Z, Y)$ are true. The atoms $connected/2$ and $nconnected/2$ state that two nodes may or may not be connected if there is an edge between them. As each probabilistic fact can be independently included or excluded from the program, there are $2^3 = 8$ possible worlds, listed in Table 1. For example, to compute the probability of the query $q = path(1, 4)$, since only w_6 and w_7 contribute to the upper bound (and none to the lower bound), $P(q) = [0, 0.06]$.

Table 1: Worlds and probabilities for Example 1. 0 and 1 indicate whether the corresponding probabilistic fact is included or excluded in the considered world, respectively. The LP/UP column indicates whether the considered world contributes to the lower and upper probability (LP), only to the upper probability (UP), or to none of them (-), for the query $path(1, 4)$.

w_{id}	$edge(1,2)$	$edge(2,4)$	$edge(1,3)$	LP/UP	Probability
w_0	0	0	0	-	0.056
w_1	0	0	1	-	0.504
w_2	0	1	0	-	0.024
w_3	0	1	1	-	0.216
w_4	1	0	0	-	0.014
w_5	1	0	1	-	0.126
w_6	1	1	0	UP	0.006
w_7	1	1	1	UP	0.054

2.2 Algebraic Model Counting

Algebraic Model Counting [25] is a framework that generalizes weighted model counting, which has recently been extended to Two-level Algebraic Model Counting (2AMC) [24]. Consider a propositional theory Π where the variables are partitioned into two subsets (V_i, V_o), two commutative semirings [22] $\mathcal{R}_i = (R^i, \oplus^i, \otimes^i, n^i_\oplus, n^i_\otimes)$ and $\mathcal{R}_o = (R^o, \oplus^o, \otimes^o, n^o_\oplus, n^o_\otimes)$, two weight functions w_i and w_o, and a transformation function f_{io} from the values of R^i to R^o. Let T be $T = (\Pi, V_i, V_o, \mathcal{R}_i, \mathcal{R}_o, w_i, w_o, f_{io})$. The 2AMC task on T is defined as:

$$2AMC(T) = \bigoplus^{o}_{I_o \in \mu(V_o)} \bigotimes^{o}_{a \in I_o} w_o(a) \otimes^o f_{io}(\bigoplus^{i}_{I_i \in \varphi(\Pi|I_o)} \bigotimes^{i}_{b \in I_i} w_i(b)) \quad (1)$$

where $\mu(V_o)$ is the set of assignments to V_o and $\varphi(\Pi \mid I_o)$ the set of assignments I_i of V_i such that (I_i, I_o) satisfies Π. If we consider a query q, inference in PASP can be represented as 2AMC [6] by considering as inner semiring $R_i = (\{(a,b) \in \mathbb{N}^2 \mid a \leq b\}, +, \cdot, (0,0), (1,1))$, outer semiring $R_o = ([0,1]^2, +, \cdot, (0,0), (1,1))$, weights w_i for the inner semiring

$$w_i(l) = \begin{cases} (0,1) & \text{if } l = not\ q \\ (1,1) & \text{otherwise} \end{cases} \tag{2}$$

weights w_o for the outer semiring ($\pi :: a$ is a probabilistic fact)

$$w_o(l) = \begin{cases} (\pi, \pi) & \text{if } l = a \text{ with } \pi :: a \\ (1-\pi, 1-\pi) & \text{if } l = not\ a \text{ with } \pi :: a \\ (1,1) & \text{otherwise} \end{cases}$$

and transformation function

$$f((v_l, v_u)) = \begin{cases} (1,1) & \text{if } v_l = v_u \text{ and } v_u > 0 \\ (0,1) & \text{if } v_u > v_l \\ (0,0) & \text{otherwise} \end{cases}$$

We are now ready to introduce our proposal.

3 Parameter Learning in Probabilistic Answer Set Programming

In this section, we first introduce the gradient semiring for PASP, then we define the parameter learning task in PASP, and finally we develop a parameter learning algorithm based on gradient descent.

3.1 The Gradient Semiring

Parameter learning requires setting the probabilities of the probabilistic facts in the program. However, some probabilistic facts may have a fixed probability. Thus, we differentiate probabilistic facts whose probabilities should be learned (we call them tunable) from the others (we call them fixed). In PLP, the gradient semiring is defined as [25] $\mathcal{R}_g = ([0,1] \times \mathbb{R}^n, \oplus, \otimes, (0, \overline{0}), (1, \overline{1}))$ where each element is a pair storing the probability value and its gradient w.r.t. the considered variable, respectively, $(a, e_a) \oplus (b, e_b) = (a + b, e_a + e_b)$, $(a, e_a) \otimes (b, e_b) = (a \cdot b, a \cdot e_b + e_a \cdot b)$ and the weight function is

$$w(l) = \begin{cases} (\pi_i, e_i) & \text{if } l = a_i \text{ with } \pi_i :: a_i \text{ tunable} \\ (1-\pi_i, -e_i) & \text{if } l = not\ a \text{ with } \pi_i :: a_i \text{ tunable} \\ (\pi_i, \overline{0}) & \text{if } l = a_i \text{ with } \pi :: a_i \text{ fixed} \\ (1-\pi_i, \overline{0}) & \text{if } l = not\ a_i \text{ with } \pi_i :: a_i \text{ fixed} \end{cases}$$

where e_i is a vector of zeros except for the position i which contains 1.

To extend this to PASP, we need a 4-tuple in the outer semiring: the first two elements are the lower and upper probability, while the remaining two are arrays of length n, where n is the number of tunable probabilistic facts, containing at position i the derivative of the query w.r.t. probabilistic fact a_i. Thus, we have a 2AMC task with the following components: as inner semiring the one for inference in PASP (cf. Section 2.2), as outer semiring $\mathcal{R}_o = ([0,1]^2 \times \mathbb{R}^n \times \mathbb{R}^n, \oplus, \otimes, (0,0,\overline{0},\overline{0}), (1,1,\overline{0},\overline{0}))$ with $(l_a, u_a, e_a^l, e_a^u) \oplus (l_b, u_b, e_b^l, e_b^u) = (l_a + l_b, u_a + u_b, e_a^l + e_b^l + e_a^u + e_b^u)$, $(l_a, u_a, e_a^l, e_a^u) \otimes (l_b, u_b, e_b^l, e_b^u) = (l_a \cdot l_b, u_a \cdot u_b, l_a \cdot e_b^l + l_b \cdot e_a^l, u_a \cdot e_b^u + u_b \cdot e_a^u)$, i.e., the doubled gradient semiring for PLP, as transformation function

$$f((v_l, v_u)) = \begin{cases} (1,1,\overline{0},\overline{0}) & \text{if } v_l = v_u \text{ and } v_u > 0 \\ (0,1,\overline{0},\overline{0}) & \text{if } v_u > v_l \\ (0,0,\overline{0},\overline{0}) & \text{otherwise} \end{cases}$$

as inner weight function Equation 2, and as outer weight function

$$w_o(l) = \begin{cases} (\pi_i, \pi_i, e_i, e_i) & \text{if } l = a \text{ with } \pi_i :: a_i \text{ tunable} \\ (1-\pi_i, 1-\pi_i, -e_i, -e_i) & \text{if } l = \mathit{not}\ a \text{ with } \pi_i :: a_i \text{ tunable} \\ (\pi_i, \pi_i, \overline{0}, \overline{0}) & \text{if } l = a \text{ with } \pi_i :: a_i \text{ fixed} \\ (1-\pi_i, 1-\pi_i, \overline{0}, \overline{0}) & \text{if } l = \mathit{not}\ a \text{ with } \pi_i :: a_i \text{ fixed} \end{cases}$$

3.2 Parameter Learning in PASP

Here, we extend the parameter learning setting of EMBLEM [7] to PASP. We provide a definition which targets the upper probability. The one for the lower probability is analogous. We minimize the mean squared error (MSE) to obtain the best probabilities for the probabilistic facts.

Definition 1. *(Parameter Learning in PASP). Given a PASP P with probabilistic facts with unknown probabilities (denote with Π the set of such probabilities), a set of ground atoms E^+ called positive examples, and a set of ground atoms E^- called negative examples, the goal of parameter learning is to find the best probabilities Π^* for the probabilistic facts such that*

$$\arg\min_{\Pi} MSE = \arg\min_{\Pi} \frac{1}{|E|}\Big(\sum_{e^+ \in E^+} (\overline{\mathrm{P}}(e^+) - 1)^2 + \sum_{e^- \in E^-} (\overline{\mathrm{P}}(e^-))^2 \Big). \qquad (3)$$

Note that there are alternative definitions for this task in PASP [2,3,4]. In this context, the probability of positive (negative) examples $\overline{\mathrm{P}}(e^+)$ ($\overline{\mathrm{P}}(e^-)$) is defined as the probability of the query given a mega-example, i.e. a set of ground atoms.

EMBLEM adopts Expectation Maximization and considers only dynamically *stratified* programs, i.e., every world has a unique stable model (which also coincides with the Well-founded model [30]). Here we go over this and consider also non-stratified programs, where each world may have more than one stable model.

We use $\#positive(I)$ and $\#negative(I)$ to indicate whether a mega-example is negative or positive, respectively, where I is the identifier of the example. Then, we use $\#atom(I, atom)$ to specify the individual atoms of the mega-example, which can be $red(N)$, $blue(N)$, or $green(N)$, to indicate the color assigned to node N. To clarify, consider the following example.

Example 2. The *bongard* dataset [15][1], derives from one of Bongard's visual concept learning problems and features a set of pictures composed of geometric shapes, such as triangles, squares, and circles, some of which are inside others. Each picture is described by a set of logic facts. For example, a picture with a triangle $o1$ inside a circle $o2$ can be described by the following facts: $triangle(o1)$, $circle(o2)$, $inside(o1, o2)$. Here, a mega-example describes a picture with different geometric shape configurations. A negative and a positive mega-example are shown in the snippet below:

```
#negative(1).
#atom(1,square(o2)).
#atom(1,square(o1)).
#atom(1,inside(o1,o2)).

#positive(2).
#atom(2,circle(o2)).
#atom(2,triangle(o1)).
#atom(2,config(o1,up)).
#atom(2,inside(o1,o2)).
```

3.3 Algorithm

We aim to minimize the MSE of the examples through gradient descent. Suppose again that we target the upper probability. If π_j is the parameter (i.e., probability) associated with the probabilistic fact a_j, the gradient of the MSE w.r.t. π_j is given by [23]

$$\frac{\partial MSE}{\partial \pi_j} = \frac{2}{|E|} \sum_{e \in E} (\overline{\mathrm{P}}(e) - P_T(e)) \cdot \frac{\partial \overline{\mathrm{P}}(e)}{\partial \pi_j}. \tag{4}$$

We can compute the probability and the gradient w.r.t. each parameter using the gradient semiring. Then, we update the parameters iteratively, as usual in gradient descent, until convergence. Our approach is summarized in Algorithm 1. First, function InitProbabilites initializes the probabilities of the probabilistic facts. Then, while the convergence criterion is not met, for each example, we apply the gradient semiring, compute the probability and the gradient w.r.t. the current parameters (function GradientSemiring), and update the loss list.

[1] Dataset available at: https://cplint.eu/example/learning/bongard.pl

Algorithm 1 PASP-GD. Parameter Learning on a PASP P with examples $E = (E^+, E^-)$, learning rate η, and convergence tolerance ϵ.

function ParameterLearning(P,E,ϵ)
  converged $\leftarrow$ *false*
  losses $\leftarrow$ [ ]
  $\Pi \leftarrow$ InitProbabilities
  $ll_0 \leftarrow -\infty$
  while not converged **do**
    for $e \in E$ **do**
      $(lp, up, \underline{e}, \overline{e}) \leftarrow$ GradientSemiring(e,P,Π)
      if target = lower **then**
        $p = lp$, $e = \underline{e}$
      else
        $p = up$, $e = \overline{e}$
      end if
      losses $\leftarrow$ *losses* $\cup\, (p - P_T(e)) \cdot e$
    end for
    $\Pi^o = \Pi - 2 \cdot \eta \cdot \mu(losses)$ ▷ μ computes the mean
    $\Pi \leftarrow$ Clip(Π^o) ▷ Clip to $[0, 1]$.
    $ll_1 \leftarrow$ ComputeLL(P,Π)
    converged $\leftarrow$ CheckConvergence(ll_0,ll_1,ϵ)
    $ll_0 \leftarrow ll_1$
  end while
end function

Lastly, we update the parameters following Equation 4 and check the convergence with function CheckConvergence.

We implemented Algorithm 1 on top of the aspmc solver [17]. We call it PASP-GD. In function GradientSemiring, each PASP obtained by considering each example is converted into a compact form (a step called knowledge compilation [13]), which is an arithmetic circuit in the form of an sd-DNNF [12]. In practice, we build this circuit only in the first iteration and store it for the following iterations. This is possible since the structure of the circuit is determined only by the structure of the program (i.e., the rules), and it is not influenced by the probabilities. In the different iterations, it is sufficient to traverse the same circuit but considering different probabilities.

4 Experiments

We conducted a series of experiments to evaluate the learning capabilities of the proposed algorithm. To ensure robustness and reduce variance in performance evaluation, we adopted 5-fold cross-validation. The learning loop stops when a convergence criterion is satisfied, defined by a tolerance threshold for the minimum required increase in the log-likelihood (LL), or when a maximum number of iterations is reached. The experiments were conducted on a machine equipped

with an AMD EPYC 9454 48-core CPU. The memory limit was set to 32 GB of RAM, and the time limit was set to one hour per run.

4.1 Datasets

We tested our algorithm (PASP-GD) on two different datasets: a non-stratified one and a stratified one. We briefly recall the definition of a stratified and non-stratified program, which is based on the definition of dependency graph. The dependency graph D_P of a program P is a directed graph [1] where each node corresponds to a ground atom in P. For every rule in the grounding of P with head atom p and body containing an atom q, the graph includes a direct edge from p to q, labelled as positive (negative) if q appears in a positive (negative) literal. A program is considered *non-stratified* if its dependency graph does not contain loops with negative edges. Such programs may have multiple or no answer sets. In contrast, *stratified* programs have no cycles through negation and have a unique answer set.

The non-stratified *coloring* dataset consists of probabilistic graphs with a total number of N nodes. Each node can be colored red, green, or blue. An example specifies the color assigned to each node and is labelled either positive or negative. Here, the edge probabilities represent the targets. This dataset was generated using Python in combination with clingo [19] to compute answer sets. An example of initial program is exemplified below.

```
red(X) :- node(X), \+ green(X), \+ blue(X).
blue(X) :- node(X), \+ green(X), \+ red(X).
green(X) :- node(X), \+ blue(X), \+ red(X).
qr :- e(A,B), red(A), red(B).
qr :- e(A,B), green(A), green(B).
qr :- e(A,B), blue(A), blue(B).
e(A,B):- edge(A,B).
e(A,B):- edge(B,A).
node(0..3).
t::edge(0,1).t::edge(0,2).t::edge(0,3).
t::edge(1,2).t::edge(1,3).t::edge(2,3).
query(qr).
```

The first three rules state that a node X must be assigned exactly one of three colors: red, green, or blue. The rules ur, ug, and ub hold if two nodes A and B connected by an edge are colored the same way. The atoms $edge(A, B)$ represent the presence of an edge between A and B for each pair of nodes $A, B \in \{0, \ldots, N-1\}$, where N is the number of nodes. The predicate $e(A, B)$ represents an undirected edge between nodes A and B, and is defined using $edge(A, B)$ and $edge(B, A)$. The probabilities of atoms for the predicate $edge(A, B)$ are tunable, and we indicate this using the letter t in place of the probability value. Note that t is used for all, but this does not state that these are related.

The examples were generated based on predefined random target values for the parameters to be learned. For each example, we generated a graph in which

each edge is included with a fixed probability. This graph was then used to produce the corresponding examples. This configuration enables the evaluation of the algorithm's efficacy by quantifying the degree of similarity between the learned parameters and the original target values. We used MSE to evaluate the distance between the learned and target probabilities, which we refer to as MSE_{LT} to differentiate it from the MSE defined in 3. We compute MSE_{LT} as follows:

$$MSE_{LT} = \frac{1}{N}\sum_{i=1}^{N}(\phi_i - t_i)^2$$

where ϕ_i and t_i are the learned and target probabilities, respectively, and N is the number of tunable parameters. Experiments were conducted with graphs comprising 4, 5, and 6 nodes. The following snippet contains two mega-examples, one positive and one negative:

```
#negative(1).
#atom(1,green(1)).
#atom(1,green(3)).
#atom(1,blue(0)).
#atom(1,red(2)).

#positive(2).
#atom(2,green(3)).
#atom(2,blue(2)).
#atom(2,red(0)).
#atom(2,red(1)).
```

We also tested the proposed algorithm on the stratified *bongard* dataset (see Example 2), to compare our approach with state-of-the-art parameter learning solvers (targeting stratified programs only), namely SLIPCOVER [8] and LIFTCOVER+ [21]. Note again that these tools do not support non-stratified programs. We considered log-likelihood (LL) and the area under the receiver operating characteristic curve (AUCROC) as metrics. Recall that the receiver operating characteristic curve plots the true positive rate versus the false positive rate to measure the model's ability to discriminate between classes. We sampled an equal number of positive and negative examples, for a total number of 50, 100, and 250. The underlying program is illustrated in the following snippet:

```
pos :- r0, circle(A), inside(B,A).
pos :- r1, circle(A), triangle(B).
t::r0.
t::r1.
query(pos).
```

The first rule states that there is a shape B inside a circle A, while the second states that there is a circle A and a triangle B. Atoms $r0$ and $r1$ are probabilistic facts whose probabilities need to be learned, denoted by t.

4.2 Results

Coloring dataset. For coloring, we monitored AUCROC and LL during training, as well as the final MSE_{LT} between learned and target probabilities. In addition, we report the LL and AUCROC on the test set. We evaluated our algorithm under various learning rate (lr) configurations (0.1, 0.5, and 0.9), with a maximum of 1000 iterations and random initialization of the parameters. As the learning rate changes, the number of iterations required for convergence also varies; however, the final results remain comparable across different configurations. In all experiments, the algorithm converged much earlier than the maximum of 1000 iterations, typically around 100.

Figure 1 presents the training AUCROC and LL. Table 2 summarizes AUCROC, LL, and MSE_{LT} obtained on the test set. Figure 1 and Table 2 report the average results across cross-validation folds for each experiment, using lr $=$ 0.5 and up to 100 iterations. The results show AUCROC scores between 0.75 and 0.89 and MSE_{LT} often below 0.1, proving that the algorithm is effective and able to produce probabilities that are very close to the target, successfully learning from the examples provided.

With $N = 5$, AUCROC ranges from 0.8240 ($e = 50$) to 0.7530 ($e = 100$) before improving again to 0.8994 ($e = 250$), while LL from -4.6855 ($e = 50$) to -11.9172 ($e = 100$) to -20.5940 ($e = 250$). A similar trend is visible for $N = 4$, but not for $N = 6$. This behavior may suggest that at $e = 100$, the model is better tuned for ranking predictions (thus optimizing AUCROC), but less well calibrated in terms of absolute probability values (LL) because the model has enough data to improve ranking performance (AUCROC) but not yet enough to fully optimize and calibrate probabilities, leading to suboptimal LL. With larger datasets ($e = 250$), the model benefits from improved calibration and overall convergence. Overall, the algorithm demonstrates a strong capacity to learn probabilistic models of graph coloring.

Table 2: Results on the coloring dataset with 5-fold cross-validation.

Nodes	Examples	Mean AUCROC	Mean LL	MSE_{LT}
4	50	0.8640	-5.1962	0.0313
4	100	0.8330	-14.5686	0.0488
4	250	0.8614	-23.4130	0.0019
5	50	0.8240	-4.6855	0.0550
5	100	0.7530	-11.9172	0.0937
5	250	0.8994	-20.5940	0.0373
6	50	0.8840	-8.0083	0.1182
6	100	0.8370	-12.2185	0.1371
6	250	0.8646	-22.9584	0.0937

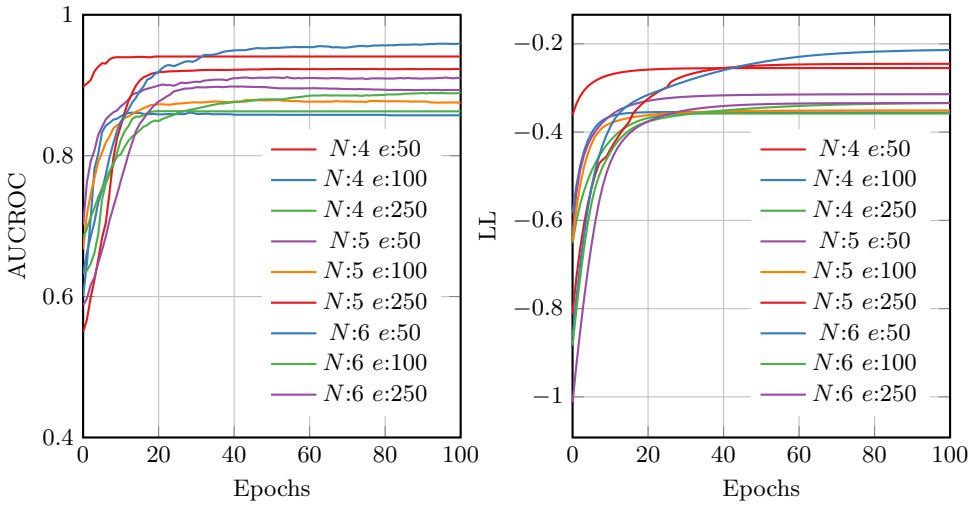

Fig. 1: Training AUCROC (left) and Training LL (right) for the coloring dataset with different settings. N and e are the number of nodes and the number of examples, respectively.

Bongard dataset. Also with the bongard dataset we experimented with various configurations, including different learning rates (0.5, 0.9, and 1), maximum number of iterations set to 100, and parameter initialization strategies (fixed at 0.5 or random). Table 3 reports the results. These changes in the parameters did not significantly affect the considered metrics as all configurations resulted in the same AUCROC with only minimal variations in the LL. This suggests that they have a negligible impact on performance. When compared to SLIPCOVER and LIFTCOVER+, our approach consistently matched or outperformed them in terms of both LL and AUCROC across all configurations. With 50 examples, all algorithms achieved a LL of approximately -3.7 and an AUCROC of 0.596. Increasing the sample size to 100 led to the highest AUCROC, though the LL slightly decreased to about -6.8. When the sample size reached 250, both LL and AUCROC showed minor declines (around -19 and 0.7, respectively), but the results remained acceptable.

Overall, this preliminary experimental evaluation demonstrated promising results on both stratified and non-stratified datasets.

5 Related Work

Parameter learning was studied in depth in the context of Probabilistic Logic Programming. An overview can be found in [28]. Among the systems, PRISM [29], EMBLEM [7], and LFI-ProbLog [18], are based on Expectation Maximization (EM), while LeProbLog [23] is based on gradient descent.

Table 3: Results on the bongard dataset with 5-fold cross-validation.

Algorithm	#Rules	#Examples	LR	Init. Prob.	Mean LL	Mean AUCROC
SLIPCOVER	2	50	-	-	-3.7938	0.5960
LIFTCOVER+	2	50	-	-	-3.7936	0.5960
PASP-GD	2	50	0.5	random	-3.7283	0.5960
PASP-GD	2	50	0.9	random	-3.7160	0.5960
PASP-GD	2	50	1.0	random	-3.7273	0.5960
PASP-GD	2	50	0.5	0.5	-3.7159	0.5960
PASP-GD	2	50	0.9	0.5	-3.7158	0.5960
PASP-GD	2	50	1.0	0.5	-3.7157	0.5960
SLIPCOVER	2	100	-	-	-6.7695	0.7390
LIFTCOVER+	2	100	-	-	-6.9500	0.7470
PASP-GD	2	100	0.5	random	-6.8277	0.7470
PASP-GD	2	100	0.9	random	-6.8278	0.7470
PASP-GD	2	100	1.0	random	-6.8278	0.7470
PASP-GD	2	100	0.9	0.5	-6.8278	0.7470
PASP-GD	2	100	0.5	0.5	-6.8277	0.7470
PASP-GD	2	100	1.0	0.5	-6.8278	0.7470
SLIPCOVER	2	250	-	-	-19.5835	0.6917
LIFTCOVER+	2	250	-	-	-19.9583	0.7000
PASP-GD	2	250	0.9	random	-19.3014	0.7112
PASP-GD	2	250	1.0	random	-19.3014	0.7112
PASP-GD	2	250	0.5	random	-19.3012	0.7112
PASP-GD	2	250	0.9	0.5	-19.3014	0.7112
PASP-GD	2	250	1.0	0.5	-19.3014	0.7112
PASP-GD	2	250	0.5	0.5	-19.3012	0.7112

LIFTCOVER+ [21] and SLIPCOVER [8] both perform parameter learning. LIFTCOVER+ implements both EM and gradient descent and learns the parameters of liftable probabilistic logic programs, i.e. a restricted class of probabilistic logic programs consisting of clauses with a single probabilistic head atom and deterministic bodies. LIFTCOVER+ derives from SLIPCOVER [8], which learns general probabilistic logic programs by searching in the space of clauses and iteratively refining the theory through the greedy inclusion of improved clause refinements. An alternative approach can be found in [5] where the authors adopted constrained optimization solvers to handle the task. However, these algorithms are limited to stratified programs and cannot be applied to non-stratified ones.

For PASP, parameter learning is handled in [2,3,4], although within a different framework. There, they consider the learning from interpretation setting (as happens in LFI-ProbLog) where examples are given as partial interpretations, i.e., set of atoms of the form $\mathcal{I} = \langle I_t, I_f \rangle$ where I_t and I_f are sets of atoms. Then, each example $\mathcal{I}$ is associated with a conjunction $q_{\mathcal{I}}$ of the form

$\bigwedge_{a \in I_t} a \bigwedge_{b \in I_f} not\ b$. The task requires finding the parameters of the programs such that $\prod_{\mathcal{I}} P(q_{\mathcal{I}})$ is maximized. This is a different framework from the one considered in this paper (which is the same as EMBLEM), and it is not clear how to convert one into the other.

6 Conclusion

In this paper, we propose the gradient semiring for Probabilistic Answer Set Programming. Our proposal extends the gradient semiring for PLP to PASP, which requires two levels of algebraic model counting. We implement it into aspmc, a state-of-the-art solver, and integrate it into a gradient descent algorithm. To assess the feasibility of our approach, we run a preliminary set of experiments on two datasets with multiple configurations (different number of examples and learning rates) showing good performance in terms of LL and AUCROC. The main focus of our algorithm is to work with non-stratified datasets, in which it has demonstrated good performance and learning capability. It has also been shown to work on stratified datasets, achieving performance comparable to or exceeding that of the current state-of-the-art.

Future works include the study of structure learning for PASP. However, this is a significantly harder task since here also the structure of the program is unknown and should be learned from data.

Acknowledgments

This work has been partially supported by Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU), by the Italian Ministry of Industrial Development (MISE) under project EI-TWIN n. F/310168/05/X56 CUP B29J24000680005. DA, AB, EG, and FR are members of the Gruppo Nazionale Calcolo Scientifico – Istituto Nazionale di Alta Matematica (GNCS-INdAM). Elisabetta Gentili contributed to this paper while attending the PhD programme in Engineering Science at the University of Ferrara, Cycle XXXVIII, with the support of a scholarship financed by the Ministerial Decree no. 351 of 9th April 2022, based on the NRRP - funded by the European Union - NextGenerationEU - Mission 4 "Education and Research", Component 1 "Enhancement of the offer of educational services: from nurseries to universities" - Investment 4.1 "Extension of the number of research doctorates and innovative doctorates for public administration and cultural heritage".

References

1. Apt, K.R., Bol, R.N.: Logic programming and negation: A survey. The Journal of Logic Programming **19**, 9–71 (1994)

2. Azzolini, D.: A constrained optimization approach to set the parameters of probabilistic answer set programs. In: Bellodi, E., Lisi, F.A., Zese, R. (eds.) Inductive Logic Programming. pp. 1–15. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-49299-0_1
3. Azzolini, D., Bellodi, E., Riguzzi, F.: Learning the parameters of probabilistic answer set programs. In: Muggleton, S.H., Tamaddoni-Nezhad, A. (eds.) Inductive Logic Programming. pp. 1–14. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-55630-2_1
4. Azzolini, D., Gentili, E., Riguzzi, F.: Symbolic parameter learning in probabilistic answer set programming. Theory and Practice of Logic Programming **24**(4), 698–715 (2024). https://doi.org/10.1017/S1471068424000334
5. Azzolini, D., Riguzzi, F.: Optimizing probabilities in probabilistic logic programs. Theory and Practice of Logic Programming **21**(5), 543–556 (2021). https://doi.org/10.1017/S1471068421000260
6. Azzolini, D., Riguzzi, F.: Inference in probabilistic answer set programming under the credal semantics. In: Basili, R., Lembo, D., Limongelli, C., Orlandini, A. (eds.) AIxIA 2023 - Advances in Artificial Intelligence. Lecture Notes in Artificial Intelligence, vol. 14318, pp. 367–380. Springer, Heidelberg, Germany (2023). https://doi.org/10.1007/978-3-031-47546-7_25
7. 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
8. 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
9. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (december 2011). https://doi.org/10.1145/2043174.2043195
10. Cozman, F.G., Mauá, D.D.: On the semantics and complexity of probabilistic logic programs. Journal of Artificial Intelligence Research **60**, 221–262 (2017). https://doi.org/10.1613/jair.5482
11. Cozman, F.G., Mauá, D.D.: The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference. International Journal of Approximate Reasoning **125**, 218–239 (2020). https://doi.org/10.1016/j.ijar.2020.07.004
12. Darwiche, A.: Decomposable negation normal form. Journal of the ACM **48**(4), 608–647 (2001). https://doi.org/doi.org/10.1145/502090.502091
13. Darwiche, A., Marquis, P.: A knowledge compilation map. Journal of Artificial Intelligence Research **17**, 229–264 (2002). https://doi.org/10.1613/jair.989
14. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: Veloso, M.M. (ed.) 20th International Joint Conference on Artificial Intelligence (IJCAI 2007). vol. 7, pp. 2462–2467. AAAI Press (2007)
15. De Raedt, L., Van Laer, W.: Inductive constraint logic. In: International Workshop on Algorithmic Learning Theory. pp. 80–94. Springer (1995)
16. Dries, A., Kimmig, A., Meert, W., Renkens, J., Van den Broeck, G., Vlasselaer, J., De Raedt, L.: Problog2: Probabilistic logic programming. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III 15. pp. 312–315. Springer (2015)

17. Eiter, T., Hecher, M., Kiesel, R.: aspmc: New frontiers of algebraic answer set counting. Artificial Intelligence **330**, 104109 (2024). https://doi.org/10.1016/j.artint.2024.104109
18. Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D.S., Gutmann, B., Thon, I., Janssens, G., De Raedt, L.: Inference and learning in probabilistic logic programs using weighted Boolean formulas. Theory and Practice of Logic Programming **15**(3), 358–401 (2015). https://doi.org/10.1017/S1471068414000076
19. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. Theory and Practice of Logic Programming **19**(1), 27–82 (2019). https://doi.org/10.1017/S1471068418000054
20. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: 5th International Conference and Symposium on Logic Programming (ICLP/SLP 1988). vol. 88, pp. 1070–1080. MIT Press, USA (1988)
21. Gentili, E., Bizzarri, A., Azzolini, D., Zese, R., Riguzzi, F.: Regularization in probabilistic inductive logic programming. In: Bellodi, E., Lisi, F.A., Zese, R. (eds.) Inductive Logic Programming. pp. 16–29. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-49299-0_2
22. Gondran, M., Minoux, M.: Graphs, Dioids and Semirings: New Models and Algorithms (Operations Research/Computer Science Interfaces Series). Springer Publishing Company, Incorporated, 1 edn. (2008)
23. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.D.: Parameter learning in probabilistic databases: A least squares approach. In: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD 2008). Lecture Notes in Computer Science, vol. 5211, pp. 473–488. Springer (2008). https://doi.org/10.1007/978-3-540-87479-9_49
24. Kiesel, R., Totis, P., Kimmig, A.: Efficient knowledge compilation beyond weighted model counting. Theory and Practice of Logic Programming **22**(4), 505–522 (2022). https://doi.org/10.1017/S147106842200014X
25. Kimmig, A., Van den Broeck, G., De Raedt, L.: Algebraic model counting. Journal of Applied Logic **22**(C), 46–62 (2017). https://doi.org/10.1016/j.jal.2016.11.031
26. Raedt, L.D., Kersting, K., Natarajan, S., Poole, D.: Statistical relational artificial intelligence: Logic, probability, and computation. Synthesis lectures on artificial intelligence and machine learning **10**(2), 1–189 (2016)
27. Richardson, M., Domingos, P.: Markov logic networks. Machine learning **62**, 107–136 (2006)
28. Riguzzi, F.: Foundations of Probabilistic Logic Programming Languages, Semantics, Inference and Learning, Second Edition. River Publisher, Gistrup, Denmark (2022)
29. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming. pp. 715–729. MIT Press (1995). https://doi.org/10.7551/mitpress/4298.003.0069
30. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. Journal of the ACM **38**(3), 620–650 (1991)