Regularization in Probabilistic Inductive Logic Programming

Regularization in Probabilistic Inductive Logic Programming

Elisabetta Gentili[1(✉)], Alice Bizzarri[1], Damiano Azzolini[2], Riccardo Zese[3], and Fabrizio Riguzzi[4]

[1] Department of Engineering, University of Ferrara, Ferrara, Italy
{elisabetta.gentili1,alice.bizzarri}@unife.it
[2] Department of Environmental and Prevention Sciences, University of Ferrara, Ferrara, Italy
damiano.azzolini@unife.it
[3] Department of Chemical, Pharmaceutical and Agricultural Sciences, University of Ferrara, Ferrara, Italy
riccardo.zese@unife.it
[4] Department of Mathematics and Computer Science, University of Ferrara, Ferrara, Italy
fabrizio.riguzzi@unife.it

Abstract. Probabilistic Logic Programming combines uncertainty and logic-based languages. Liftable Probabilistic Logic Programs have been recently proposed to perform inference in a lifted way. LIFTCOVER is an algorithm used to perform parameter and structure learning of liftable probabilistic logic programs. In particular, it performs parameter learning via Expectation Maximization and LBFGS. In this paper, we present an updated version of LIFTCOVER, called LIFTCOVER+, in which regularization was added to improve the quality of the solutions and LBFGS was replaced by gradient descent. We tested LIFTCOVER+ on the same 12 datasets on which LIFTCOVER was tested and compared the performances in terms of AUC-ROC, AUC-PR, and execution times. Results show that in most cases Expectation Maximization with regularization improves the quality of the solutions.

Keywords: Probabilistic Inductive Logic Programming · Regularization · Statistical Relational Artificial Intelligence

1 Introduction

Probabilistic Logic Programming (PLP) combines uncertainty and logic-based languages [17]. Given its expressiveness, in the last decades PLP, and in particular PLP under the distribution semantics [21], has been widely adopted in domains characterized by uncertainty [5,11,12,19,20]. A probabilistic logic program without function symbols under the distribution semantics defines a probability distribution over normal logic programs, also called *instances* or *worlds*.

E. Bellodi et al. (Eds.): ILP 2023, LNAI 14363, pp. 16–29, 2023.
https://doi.org/10.1007/978-3-031-49299-0_2

The distribution is extended to a joint distribution over worlds and interpretations (or queries) and the probability of a query can be obtained from this distribution [17]. Logic Programs with Annotated Disjunctions (LPADs) [26] are a PLP language under the distribution semantics. In LPADs without function symbols, heads of clauses are disjunctions in which each atom is annotated with a probability. Since learning probabilistic logic programs is expensive, various approaches have been proposed to overcome this problem. Lifted inference [15] was introduced to improve the performances of reasoning in probabilistic relational models by taking into consideration populations of individuals instead of considering each individual separately. Liftable Probabilistic Logic Programs have been recently proposed to perform inference in a lifted way. LIFTCOVER [13] is an algorithm that performs structure and parameter learning (via Expectation-Maximization (LIFTCOVER-EM) or Limited-memory BFGS (LIFTCOVER-LBFGS)) of liftable probabilistic logic programs. Previous results [13] showed that LIFTCOVER-EM often outperformed LIFTCOVER-LBFGS and other systems at the state of the art. In this paper, we present LIFTCOVER+, an algorithm that extends LIFTCOVER with regularization and gradient descent for parameter learning to improve the quality of the solutions and prevent overfitting. We test LIFTCOVER+ on 12 real-world datasets and compare the results with LIFTCOVER-EM. Empirical results show that LIFTCOVER+ with the regularized Expectation-Maximization algorithm allows to obtain slightly better results than the original LIFTCOVER-EM.

The paper is organized as follows: Sect. 2 presents background on PLP; Sect. 3 introduces LIFTCOVER+; Sect. 4 shows the results of the experiments; in Sect. 5 we discuss related work; and in Sect. 6 we draw the conclusions.

2 Background

We consider the liftable PLP language [13], a restriction of probabilistic logic programs so that inference can be performed in a lifted way. Such programs contain clauses with a single annotated atom in the head and the predicate of this atom is the same for all clauses, i.e., clauses of the form:

$$C_i = h_i : \Pi_i :- b_{i1}, \ldots, b_{iu_i}$$

where the single atom in the head is built over predicate *target/a*, with a the arity. The bodies of the clauses contain other predicates than *target/a* and their facts and rules have a single atom in the head with probability 1 (they are certain). The predicate *target/a* is the target of learning and the other predicates are *input predicates*. In other words, in the liftable PLP language uncertainty appears only in the rules. The goal is to compute the probability of a ground instantiation (or query) q of *target/a*. To do so, we find the number of ground instantiations of clauses for *target/a* such that the body is true and the head is equal to q. Let $\{\theta_{i1}, ..., \theta_{im_i}\}$ be the m_i instantiations for clause C_i, $i = 1, ..., n$. Every instantiation θ_{ij} corresponds to a random variable X_{ij} that is equal to 1 (0) with probability Π_i $(1 - \Pi_i)$. The query q is true if at least one random

variable for a rule is true, i.e., takes value 1. Equivalently, the query q is false only if none of the random variables is true. Since all the random variables are mutually independent the probability that q is true can be computed as $P(q) = 1 - \prod_{i=1}^{n}(1 - \Pi_i)^{m_i}$. The fact that the random variables associated to the rules are mutually independent does not limit the capability to represent probability distributions, as shown in [17].

LIFTCOVER [13], shown in Algorithm 1, learns the structure of liftable probabilistic logic programs. Given a set $E^+ = \{e_1, \ldots, e_Q\}$ of positive examples, a set $E^- = \{e_{Q+1}, \ldots, e_R\}$ of negative examples, and a background knowledge B (possibly a normal logic program defining the input predicates), the goal of structure learning is to find a liftable probabilistic logic program T such that the likelihood

$$L = \prod_{q=1}^{Q} P(e_q) \prod_{r=Q+1}^{R} P(\neg e_r)$$

is maximized. LIFTCOVER solves this problem by first identifying good clauses guided by the log-likelihood (LL) of the data, with a top-down beam search. The refinement operator adds a literal taken from a bottom clause to the body of the current clause. The beam search is repeated a user-defined number of times or until the beam is empty. Then, parameter learning is performed on the full set of clauses found, which is considered as a single theory. LIFTCOVER can use either Expectation-Maximization (EM) or Limited-memory BFGS (LBFGS). LBFGS is used to find the values of the parameters that optimize the likelihood by exploiting the gradient of the log-likelihood with respect to the parameters. The likelihood can be unfolded to

$$L = \prod_{l=1}^{n}(1 - \Pi_l)^{m_{l-}} \prod_{q=1}^{Q}\left(1 - \prod_{l=1}^{n}(1 - \Pi_l)^{m_{lq}}\right)$$

where m_{iq} (m_{ir}) is the number of instantiations of C_i whose head is e_q (e_r) and whose body is true, and $m_{l-} = \sum_{r=Q+1}^{R} m_{lr}$. Its gradient can be computed as:

$$\frac{\partial L}{\partial \Pi_i} = \frac{L}{1 - \Pi_i}\left(\sum_{q=1}^{Q} m_{iq}\left(\frac{1}{P(e_q)} - 1\right) - m_{i-}\right) \tag{1}$$

Because the equation $\frac{\partial L}{\partial \Pi_i} = 0$ does not admit a closed-form solution, optimization is needed to find the maximum of L. The clauses with a probability below a user-defined threshold are discarded.

In models in which the variables are hidden, the EM algorithm [7] must be used to find the maximum likelihood estimates of parameters. In the Expectation step, the distribution of the unseen variables in each instance is computed given the observed data and the current value of the parameters. In the Maximization step, the new parameters are computed so that the expected likelihood is maximized. The alternation between the Expectation and the Maximization steps continues until the likelihood does not improve anymore.

To use the EM algorithm, the distribution of the hidden variables given the observed ones, $P(X_{ij} = 1|e)$ and $P(X_{ij} = 1|\neg e)$ has to be computed. Given that $P(X_{ij} = 1, e) = P(e|X_{ij} = 1) \cdot P(X_{ij} = 1) = P(X_{ij} = 1) = \Pi_i$ and $P(e|X_{ij} = 1) = 1$,

$$P(X_{ij} = 1|e) = \frac{P(X_{ij} = 1, e)}{P(e)} = \frac{\Pi_i}{1 - \prod_{i=1}^{n}(1 - \Pi_i)^{m_i}} \tag{2}$$

$$P(X_{ij} = 0|e) = 1 - \frac{\Pi_i}{1 - \prod_{i=1}^{n}(1 - \Pi_i)^{m_i}} \tag{3}$$

Since $P(X_{ij} = 1, \neg e) = P(\neg e|X_{ij} = 1) \cdot P(X_{ij} = 1) = 0$ and $P(\neg e|X_{ij} = 1) = 0$,

$$P(X_{ij} = 1|\neg e) = 0 \tag{4}$$

$$P(X_{ij} = 0|\neg e) = 1 \tag{5}$$

3 LIFTCOVER+

LIFTCOVER can learn very large sets of clauses that may overfit the data. For this reason, we introduce LIFTCOVER+, a modified version of LIFTCOVER that adds regularization to perform parameter learning and uses gradient descent instead of LBFGS to optimize the likelihood.

Regularization is a well-known technique to prevent overfitting, in which a penalty term is added to the loss function to penalize large weights. In this way, we aim to obtain few clauses with large weights. Clauses with small weights have little influence on the probability of the query and can be removed, thus simplifying the theory. Regularization is usually performed in gradient-based algorithms, but it can be performed in EM as well in the Maximization phase, where the parameters that maximized the LL are found. For EM, regularization can be Bayesian, L1, or L2.

In Bayesian regularization, the parameters are updated assuming a prior distribution that takes the form of a Dirichlet probability density with parameters $[a, b]$. It has the same effect as having observed a extra occurrences of $X_{ij} = 1$ and b extra occurrences of $X_{ij} = 0$. If b is much larger than a, this has the effect to shrink the parameters. L1 and L2 differ in how they penalize the loss function: L1 adds the sum of the absolute value of the parameters to the loss function while L2 adds the sum of their squares.

The L1 objective function [14] is:

$$J_1(\theta) = N_1 \cdot log\theta + N_0 \cdot log(1 - \theta) - \gamma\theta \tag{6}$$

where $\theta = \pi_i$, N_0 and N_1 are the expected occurrences of $X_{ij} = 0$ and $X_{ij} = 1$ computed in the Expectation step, and γ is the regularization coefficient. The value of θ that maximizes J_1 is computed in the Maximization step by solving the equation $\frac{\partial J(\theta)}{\partial \theta} = 0$ [14]. $J_1(\theta)$ is maximum at

$$\theta_1 = \frac{4N_1}{2(\gamma + N_0 + N_1 + \sqrt{(N_0 + N_1)^2 + \gamma^2 + 2\gamma(N_0 - N_1)})} \tag{7}$$

The L2 objective function [14] is:

$$J_2(\theta) = N_1 \cdot log\theta + N_0 \cdot log(1-\theta) - \frac{\gamma}{2}\theta^2 \tag{8}$$

and value of θ that maximizes J_2, is:

$$\theta_2 = \frac{2\sqrt{\frac{3N_0+3N_1+\gamma}{\gamma}}\cos\left(\frac{\arccos\left(\frac{\sqrt{\frac{\gamma}{3N_0+3N_1+\gamma}}\left(\frac{9N_0}{2}-9N_1+\gamma\right)}{3N_0+3N_1+\gamma}\right)}{3} - \frac{2\pi}{3}\right)}{3} + \frac{1}{3} \tag{9}$$

In LIFTCOVER+, LBFGS is replaced by regularized gradient descent. The objective function is the sum of cross entropy errors err_i for all the examples:

$$err = \sum_{i=1}^{Q+R} (-y_i \log P(e_i) - (1-y_i)\log(1-P(e_i))) \tag{10}$$

where $Q+R$ is the total number of examples, e_i is an example, and y_i is its sign, thus y_i equals to 1 (0) if the example is positive (negative). L1 regularization can then be applied to minimize the loss function [14]:

$$err_{L1} = \sum_{i=1}^{Q+R} -y_i \cdot logP(e_i) - (1-y_i) \cdot log(1-P(e_i)) + \gamma \sum_{i=1}^{k} |\pi_i| \tag{11}$$

where k is the number of parameters and the π_is are the probabilities of the clauses. After learning the parameters, all the clauses with a probability below a fixed threshold are removed.

4 Experiments

The main goal of the experiments is to assess whether adding regularization to LIFTCOVER+ improves the quality of the solution. All experiments were conducted on a GNU/Linux machine with an Intel Core i3-10320 Quad Core 3.80 GHz CPU.

We tested LIFTCOVER+ on 12 real-world datasets: UW-CSE [10] (a dataset that describes the Computer Science Department of the University of Washington, used to predict the fact that a student is advised by a professor), Mondial [22] (a dataset containing information regarding geographical regions of the world, such as population size, political system, and the country border relationship), Carcinogenesis [23] (a classic ILP benchmark dataset for Quantitative Structure-Activity Relationship (QSAR), i.e., predicting the biological activity of chemicals from their physicochemical properties or molecular structure. The goal is to predict the carcinogenicity of compounds from their chemical structure), Mutagenesis [24] (a classic ILP benchmark dataset for QSAR in which the goal is to predict the mutagenicity (a property correlated with carcinogenicity) of compounds from their chemical structure), Bupa (for diagnosing patients

Algorithm 1. Function LIFTCOVER

```
function LIFTCOVER(NB, NI, NInt, NS, NA, NV)
    Beam =InitialBeam(NInt, NS, NA)                          ▷ Bottom clauses building
    CC ← ∅
    Steps ← 1
    NewBeam ← []
    repeat
        Remove the first couple ((Cl, Literals), LL) from Beam          ▷ Remove the first clause
        Refs ←ClauseRefinements((Cl, Literals, NV))          ▷ Find all refinements Refs of (Cl, Literals)
        for all (Cl', Literals') ∈ Refs do
            (LL'', {Cl''}) ←LearnWeights(I, {Cl'})
            NewBeam ←Insert((Cl'', Literals'), LL'', NewBeam, NB)   ▷ The refinement is inserted in the beam in order of likelihood, possibly removing the last clause if the size of the beam NB is exceeded
            CC ← CC ∪ {Cl'}
        end for
        Beam ← NewBeam
        Steps ← Steps + 1
    until Steps > NI or Beam is empty
    (LL, Th) ←LearnWeights(CC)
    Remove from Th the clauses with a weight smaller than WMin
    return Th
end function
```

with liver disorders), Nba (for predicting the results of NBA basketball games), Pyrimidine and Triazine[1] (QSAR datasets for predicting the inhibition of dihydrofolate reductase by pyrimidines and triazines, respectively), Financial (for predicting the success of loan applications by clients of a bank), Sisya and Sisyb (datasets regarding insurance business clients, used to classify households and persons in relation to private life insurance), and Yeast (for predicting if a yeast gene codes for a protein involved in metabolism) from [25][2]. Table 1 shows the characteristics of the datasets.

Four different configurations of LIFTCOVER+ were compared: EM with Bayesian regularization (EM-Bayes), EM with L1 regularization (EM-L1), EM with L2 regularization (EM-L2), and gradient descent with fixed learning rate $\eta = 0.0001$ and L1 regularization (GD). Hyper-parameters for Bayesian regularization were set as $a = 0$ and b equal to 15% of the total number of examples in the dataset. We set $\gamma = 50$ for L1 and L2 in EM, and $\gamma = 10$ for L1 in gradient descent. The parameters controlling structure learning are the following: *NInt* is the number of mega-examples on which to build the bottom clauses, *NA* is the number of bottom clauses to be built for each mega-example, *NS* is the number of saturation steps (for building the bottom clauses), *NI* is the maximum number of clause search iterations, the size *NB* of the beam, *NV* is the maximum number of variables in a rule, and *WMin* is the minimum probability under which the rule is removed. Their values are listed in Table 2.

All the configurations were evaluated in terms of Area Under the Precision-Recall Curve (AUC-PR) and Area Under the Receiver Operating Characteristics

[1] https://relational.fit.cvut.cz/.
[2] https://dtai.cs.kuleuven.be/ACE/doc/.

Curve (AUC-ROC). Both were computed with the methods reported in [4,8]. LIFTOCOVER+ was compared with LIFTCOVER-EM from [13].

Table 1. Characteristics of the datasets for the experiments: number of predicates (P), of tuples (T) (i.e., ground atoms), of positive (PEx) and negative (NEx) examples for target predicate(s), of folds (F). The number of tuples includes the target positive examples.

Dataset	P	T	PEx	NEx	F
Financial	9	92658	34	223	10
Bupa	12	2781	145	200	5
Mondial	11	10985	572	616	5
Mutagen	20	15249	125	126	10
Sisyb	9	354507	3705	9229	10
Sisya	9	358839	10723	6544	10
Pyrimidine	29	2037	20	20	4
Yeast	12	53988	1299	5456	10
Nba	4	1218	15	15	5
Triazine	62	10079	20	20	4
UW-CSE	15	2673	113	20680	5
Carcinogen	36	24533	182	155	1

Tables 3, 4, and 5 show the performances of the different configurations in terms of average over the folds of AUC-ROC, AUC-PR, and the execution times, respectively. The results of LIFTCOVER-EM were taken from [13]. Execution time for LIFTCOVER+ was scaled (i.e., multiplied by 3.8/2.4) in order to compare them with those of LIFTCOVER-EM in [13] that were executed on a machine with Intel Xeon Haswell E5-2630 v3 (2.40GHz) CPU. Figures 1, 2, and 3 show the histograms of the above-mentioned data.

LIFTCOVER+ performs slightly better than LIFTCOVER-EM in terms of AUC-PR on 6 datasets out of 12 with EM-BAYES and EM-L1, on 7 datasets with EM-L2, and on 3 datasets with GD. As a matter of fact, the average AUC-PR over all datasets is higher for LIFTCOVER+ with EM and L2 regularization, followed closely by Bayesian regularization. Results obtained with LIFTCOVER+ and GD were considerably worse on the Pyramidine and Yeast datasets and were lower in almost all other cases. In particular, LIFTCOVER+ was able to significantly improve the performance on the Nba dataset achieving an AUC-PR of 0.7 against 0.5 reached by LIFTCOVER-EM. Despite that, the Sisyb dataset seems to remain a challenge for LIFTCOVER+ (both with EM and GD). Regarding AUC-ROC, LIFTCOVER+ beats LIFTCOVER-EM on 4 datasets out of 12 with EM-Bayes and EM-L1, on 3 datasets with EM-L2, and on 2 datasets with GD. In general, GD led to a deterioration of the solution in

Table 2. Parameters controlling structure search for LIFTCOVER+.

Dataset	*NB*	*NI*	*NInt*	*NS*	*NA*	*NV*	*WMin*
Financial	100	20	16	1	1	4	1e-4
Bupa	100	20	4	1	1	4	1e-4
Mondial	1000	10	1	2	6	5	1e-4
Mutagen	100	10	4	1	1	4	1e-4
Sisyb	100	20	10	1	1	50	1e-4
Sisya	100	20	4	1	1	4	1e-4
Pyrimidine	100	20	4	1	1	100	1e-4
Yeast	100	20	12	1	1	4	1e-4
Nba	100	20	4	1	1	100	1e-4
Triazine	100	20	4	1	1	4	1e-4
UW-CSE	100	60	4	1	4	4	1e-4
Carcinogen	100	60	16	2	1	3	1e-4

most cases, probably because the loss function is highly non-convex and GD ends up in local minima, while EM seems more capable of escaping local minima. In terms of execution times, LIFTCOVER+ is comparable to LIFTCOVER-EM, although it was slower in some cases. This is especially true for GD, which on some datasets (Bupa, Mondial, Mutagenesis, Pyramidine, Yeast, Triazine, Carcinogenesis) turns out to be slower by one or more orders of magnitude. However, it must be noted that the scaling approach we have used is only a rough approximation, as the architecture of the two processors is different and thus differences in caches and pipelining may have an effect. In the future, we plan to repeat the LIFTCOVER+ experiments on a machine more similar to the one of LIFTCOVER-EM.

5 Related Work

Lifted inference for PLP under the distribution semantics has been surveyed in [18], in which the authors describe and evaluate three different approaches, namely Lifted Probabilistic Logic Programming (LP^2), lifted inference with aggregation parfactors, and Weighted First Order Model Counting (WFOMC). The authors of [9], instead, focused their survey on lifted graphical models.

LIFTCOVER (and thus LIFTCOVER+) derives from SLIPCOVER [3], an algorithm for learning general PLP by performing a search in the space of clauses and then refining it by greedily adding refined clauses into the theory. Aside from the simplified structure search, LIFTCOVER and LIFTCOVER+ differ from SLIPCOVER also in the approach used for parameter learning. While SLIPCOVER uses EMBLEM [2] to learn the parameters of a probabilistic logic program by applying EM over Binary Decision Diagrams [1], LIFTCOVER and LIFTCOVER+ use EM, LBFGS, and gradient descent.

Table 3. Average AUC-ROC over the datasets for each configuration: EM with Bayes regularization (EM-Bayes), EM with L1 regularization (EM-L1), EM with L2 regularization (EM-L2), and gradient descent (GD). For each row, the best result is highlighted in bold.

Dataset	EM-Bayes	EM-L1	EM-L2	GD	LIFTCOVER-EM
Financial	0.521	0.459	**0.528**	0.389	0.432
Bupa	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Mondial	0.586	0.547	0.572	0.528	**0.663**
Mutagen	0.934	0.918	**0.939**	0.594	0.931
Sisyb	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**
Sisya	**0.720**	**0.720**	**0.720**	**0.720**	0.372
Pyrimidine	0.975	0.880	0.910	0.160	**1.000**
Yeast	0.785	0.783	0.785	0.530	**0.786**
Nba	**0.725**	**0.725**	**0.725**	0.675	0.531
Triazine	0.425	0.390	0.430	0.580	**0.713**
UW-CSE	0.976	**0.977**	0.975	0.951	**0.977**
Carcinogen	0.720	0.692	0.687	0.500	**0.766**
Average	***0.739***	*0.716*	*0.731*	*0.594*	*0.723*

Table 4. Average AUC-PR over the datasets for each configuration: EM with Bayes regularization (EM-Bayes), EM with L1 regularization (EM-L1), EM with L2 regularization (EM-L2), and gradient descent (GD). For each row, the best result is highlighted in bold.

Dataset	EM-Bayes	EM-L1	EM-L2	GD	LIFTCOVER-EM
Financial	0.155	0.127	**0.169**	0.124	0.126
Bupa	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Mondial	0.717	0.739	0.743	0.712	**0.763**
Mutagen	0.966	0.964	**0.971**	0.759	**0.971**
Sisyb	**0.286**	**0.286**	**0.286**	**0.286**	**0.286**
Sisya	**0.708**	**0.708**	**0.708**	**0.708**	0.706
Pyrimidine	0.988	0.913	0.947	0.378	**1.000**
Yeast	0.499	0.486	0.497	0.242	**0.502**
Nba	**0.789**	**0.789**	**0.789**	0.743	0.550
Triazine	0.452	0.430	0.463	0.617	**0.734**
UW-CSE	0.341	**0.358**	0.339	0.158	0.220
Carcinogen	0.687	0.691	**0.722**	0.513	0.672
Average	*0.632*	*0.624*	***0.636***	*0.520*	*0.628*

Table 5. Average time in seconds over the datasets for each configuration: EM with Bayes regularization (EM-Bayes), EM with L1 regularization (EM-L1), EM with L2 regularization (EM-L2), and gradient descent (GD). For each row, the best result is highlighted in bold.

Dataset	EM-Bayes	EM-L1	EM-L2	GD	LIFTCOVER-EM
Financial	0.280	0.275	0.278	2.360	**0.235**
Bupa	0.246	0.244	0.244	510.206	**0.243**
Mondial	6.480	**4.841**	5.149	299.875	5.911
Mutagen	15.073	**10.796**	15.653	266.196	12.770
Sisyb	0.232	0.231	0.233	0.490	**0.226**
Sisya	1.117	1.108	1.111	**0.656**	0.932
Pyrimidine	44.712	**23.408**	25.766	266.310	54.990
Yeast	60.143	52.503	54.288	994.811	**0.502**
Nba	0.658	0.705	0.624	0.913	**0.599**
Triazine	33.648	**23.871**	30.305	276.304	56.690
UW-CSE	74.163	72.961	74.456	75.656	**8.054**
Carcinogen	14.483	16.826	13.458	778.596	**7.850**
Average	*20.936*	*17.314*	*18.464*	*289.364*	***12.417***

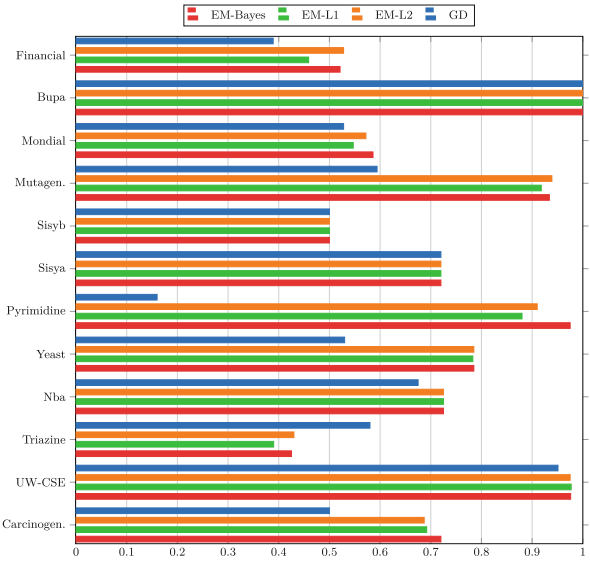

Fig. 1. Histograms of average AUC-ROC over the datasets for each configuration: EM with Bayes regularization (EM-Bayes), EM with L1 regularization (EM-L1), EM with L2 regularization (EM-L2), and gradient descent (GD).

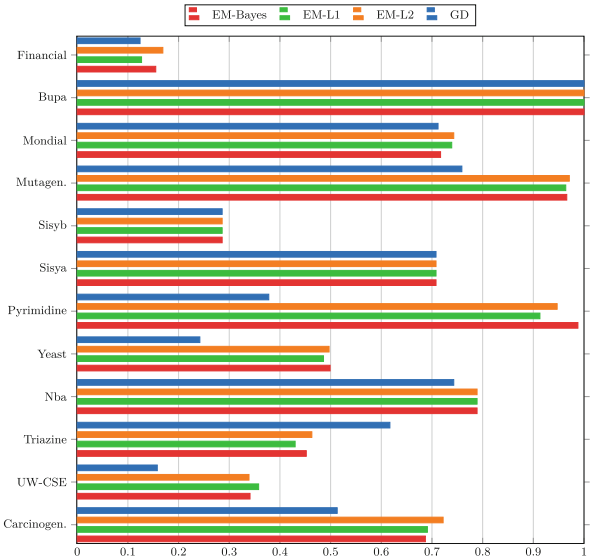

Fig. 2. Histograms of average AUC-PR over the datasets for each configuration: EM with Bayes regularization (EM-Bayes), EM with L1 regularization (EM-L1), EM with L2 regularization (EM-L2), and gradient descent (GD).

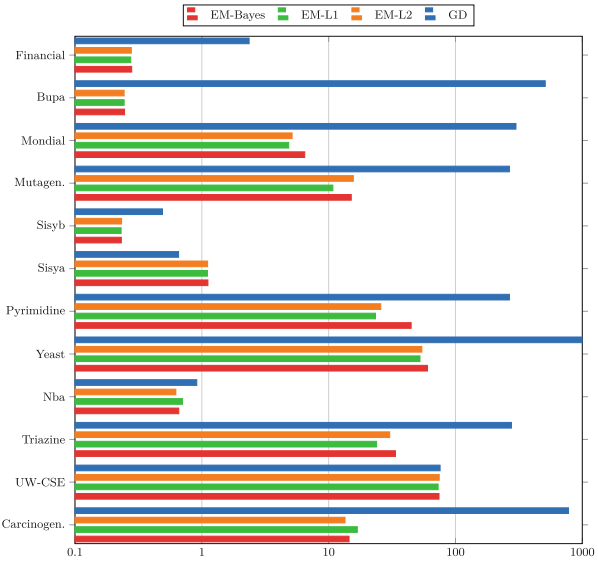

Fig. 3. Histograms of average time in seconds over the datasets for each configuration: EM with Bayes regularization (EM-Bayes), EM with L1 regularization (EM-L1), EM with L2 regularization (EM-L2), and gradient descent (GD). The scale of the X axis is logarithmic.

Hierarchical PLP (HPLP) [14] is a restriction of the general PLP language in which clauses and predicates are hierarchically organized. HPLPs can be efficiently converted into arithmetic circuits (ACs) or deep neural networks so that inference is much cheaper than for general PLP. Liftable PLP can be seen as a restriction of HPLP. For this reason, LIFTCOVER+ is related to Liftable PLP tools such as PHIL and SLEAHP [14]. PHIL performs parameter learning of hierarchical probabilistic logic programs using gradient descent (DPHIL) or EM (EMPHIL). First, it converts the program into a set of ACs sharing parameters. Then, it applies gradient descent or EM over the ACs, evaluating them bottom-up. On the other hand, SLEAHP learns both the structure and the parameters of HPLPs from data. It generates a large hierarchical logic program from an initial set of bottom clauses generated from a language bias [3]. Then, it applies a regularized version of PHIL to prune the initial large program by removing irrelevant rules, i.e., those for which the parameters are close to 0.

LIFTCOVER+ is related also to PROBFOIL+ [16], an algorithm used to perform parameter and structure learning of ProbLog [6] programs with a hill climbing search in the space of programs, consisting of a covering loop that adds one rule to the theory at each iteration and stops when a condition based on a global scoring function is satisfied. The rule to add is obtained from a clause search loop that builds the rule by iteratively adding literals to the body using a local scoring function as the heuristic.

6 Conclusions

In this paper, we have presented LIFTCOVER+, an updated version of LIFTCOVER that performs parameter learning using the EM algorithm or gradient descent with regularization to penalize large weights and prevent overfitting. Experiments were conducted on 12 real-world datasets and results were compared with LIFTCOVER-EM. In summary, we found that using gradient descent does not bring much benefit, having AUC-PR and AUC-ROC comparable to LIFTCOVER-EM, and execution times often much higher. On the other hand, using EM with regularization (and with L2 or Bayesian regularization especially) we obtain a higher AUC-PR on several datasets with roughly equal execution times. Furthermore, when there are no improvements, there is not a significant degradation in the quality of the solutions either. In conclusion, the present findings confirm that adding regularization can help improve the solution in terms of AUC-PR, although some datasets remain hard for LIFTCOVER+.

As future work, we plan to employ LIFTCOVER+ to learn theories from Knowledge Graphs (KG) to perform KG completion and triple classification.

Acknowledgements. This work has been partially supported by the 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 TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No. 952215, and by the "National Group of Computing Science (GNCS-INDAM)".

References

1. Akers, S.B.: Binary decision diagrams. IEEE Trans. Comput. **27**(6), 509–516 (1978)
2. Bellodi, E., Riguzzi, F.: Expectation maximization over binary decision diagrams for probabilistic logic programs. Intell. Data Anal. **17**(2), 343–363 (2013)
3. Bellodi, E., Riguzzi, F.: Structure learning of probabilistic logic programs by searching the clause space. Theory Pract. Logic Program. **15**(2), 169–212 (2015). https://doi.org/10.1017/S1471068413000689
4. Davis, J., Goadrich, M.: The relationship between precision-recall and ROC curves. In: European Conference on Machine Learning (ECML 2006), pp. 233–240. ACM (2006)
5. De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Mach. Learn. **100**(1), 5–47 (2015). https://doi.org/10.1007/s10994-015-5494-z
6. De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, pp. 2468–2473. Morgan Kaufmann Publishers Inc., San Francisco (2007)
7. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Stat. Soc. B **39**, 1–38 (1977)
8. Fawcett, T.: An introduction to ROC analysis. Pattern Recogn. Lett. **27**, 861–874 (2006)
9. Kimmig, A., Mihalkova, L., Getoor, L.: Lifted graphical models: a survey. Mach. Learn. **99**, 1–45 (2015)
10. Kok, S., Domingos, P.: Learning the structure of Markov Logic Networks. In: 22nd International Conference on Machine learning, pp. 441–448. ACM (2005)
11. Mørk, S., Holmes, I.: Evaluating bacterial gene-finding hmm structures as probabilistic logic programs. Bioinformatics **28**(5), 636–642 (2012)
12. Fadja, A.N., Riguzzi, F.: Probabilistic logic programming in action. In: Holzinger, A., Goebel, R., Ferri, M., Palade, V. (eds.) Towards Integrative Machine Learning and Knowledge Extraction. LNCS (LNAI), vol. 10344, pp. 89–116. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69775-8_5
13. Nguembang Fadja, A., Riguzzi, F.: Lifted discriminative learning of probabilistic logic programs. Mach. Learn. **108**(7), 1111–1135 (2019)
14. Nguembang Fadja, A., Riguzzi, F., Lamma, E.: Learning hierarchical probabilistic logic programs. Mach. Learn. **110**(7), 1637–1693 (2021). https://doi.org/10.1007/s10994-021-06016-4
15. Poole, D.: First-order probabilistic inference. In: Gottlob, G., Walsh, T. (eds.) IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, 9–15 August 2003, pp. 985–991. Morgan Kaufmann Publishers (2003)
16. Raedt, L.D., Dries, A., Thon, I., den Broeck, G.V., Verbeke, M.: Inducing probabilistic relational rules from probabilistic examples. In: Yang, Q., Wooldridge, M. (eds.) 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 1835–1843. AAAI Press (2015)
17. Riguzzi, F.: Foundations of Probabilistic Logic Programming Languages, Semantics, Inference and Learning, 2nd edn. River Publishers, Gistrup (2023)
18. Riguzzi, F., Bellodi, E., Zese, R., Cota, G., Lamma, E.: A survey of lifted inference approaches for probabilistic logic programming under the distribution semantics. Int. J. Approx. Reason. **80**, 313–333 (2017). https://doi.org/10.1016/j.ijar.2016.10.002

19. Riguzzi, F., Lamma, E., Alberti, M., Bellodi, E., Zese, R., Cota, G.: Probabilistic logic programming for natural language processing. In: Chesani, F., Mello, P., Milano, M. (eds.) Workshop on Deep Understanding and Reasoning, URANIA 2016. CEUR Workshop Proceedings, vol. 1802, pp. 30–37. Sun SITE Central Europe (2017)
20. Riguzzi, F., Swift, T.: Probabilistic logic programming under the distribution semantics. In: Kifer, M., Liu, Y.A. (eds.) Declarative Logic Programming: Theory, Systems, and Applications. Association for Computing Machinery and Morgan & Claypool (2018)
21. 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, Tokyo, Japan, 13–16 June 1995, pp. 715–729. MIT Press (1995). https://doi.org/10.7551/mitpress/4298.003.0069
22. Schulte, O., Khosravi, H.: Learning graphical models for relational data via lattice search. Mach. Learn. **88**(3), 331–368 (2012)
23. Srinivasan, A., King, R.D., Muggleton, S., Sternberg, M.J.E.: Carcinogenesis predictions using ILP. In: Lavrac, N., Džeroski, S. (eds.) 7th International Workshop on Inductive Logic Programming. Lecture Notes in Computer Science, vol. 1297, pp. 273–287. Springer, Berlin Heidelberg (1997)
24. Srinivasan, A., Muggleton, S., Sternberg, M.J.E., King, R.D.: Theories for mutagenicity: a study in first-order and feature-based induction. Artif. Intell. **85**(1–2), 277–299 (1996)
25. Struyf, J., Davis, J., Page, D.: An efficient approximation to lookahead in relational learners. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 775–782. Springer, Heidelberg (2006). https://doi.org/10.1007/11871842_79
26. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 431–445. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27775-0_30