Lifted Discriminative Learning of Probabilistic Logic Programs

Arnaud Nguembang Fadja[1] and Fabrizio Riguzzi[2]

[1] Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[2] Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[fabrizio.riguzzi,arnaud.nguembangfadja]@unife.it

Abstract. Probabilistic logic programming (PLP) provides a powerful tool for reasoning with uncertain relational models. However, learning probabilistic logic programs is expensive due to the high cost of inference. Among the proposals to overcome this problem, one of the most promising is lifted inference. In this paper we consider PLP models that are amenable to lifted inference and present an algorithm for performing parameter and structure learning of these models from positive and negative examples. We discuss parameter learning with EM and LBFGS and structure learning with LIFTCOVER, an algorithm similar to SLIPCOVER. The results of the comparison of LIFTCOVER with SLIPCOVER on 12 datasets show that it can achieve solutions of similar or better quality in a fraction of the time.

Keywords: Statistical Relational Learning, Probabilistic Inductive Logic Programming, Probabilistic Logic Programming, Lifted Inference, Expectation Maximization

1 Introduction

Probabilistic logic programming (PLP) is a powerful tool for reasoning in uncertain relational domains that is gaining popularity in Statistical Relational Artificial Intelligence due to its expressiveness and intuitiveness. PLP has successfully been applied in a variety of fields, such as natural language processing, information extraction and bioinformatics [18,1].

However, PLP usually require expensive learning procedures due to the high cost of inference. SLIPCOVER [3] for example performs structure learning of probabilistic logic programs using knowledge compilation for parameter learning: the expectations needed for the EM parameter learning algorithm are computed on the Binary Decision Diagrams (BDDs) that are built for inference. Compiling explanations for queries into BDDs has a #P cost.

Lifted inference [10] was proposed for improving the performance of reasoning in probabilistic relational models by reasoning on whole populations of individuals instead of considering each individual separately. For example, consider the following Logic Program with Annotated Disjunctions (adapted from [5]):

$popular(X) : p :- friends(X, Y), famous(Y).$
where $p \in [0, 1]$. In this case $P(popular(john)) = 1 - (1 - p)^n$ where n is the number of famous friends of $john$. Computing this value has a cost logarithmic in n, as computing a^b is $\Theta(\log b)$ with the "square and multiply" algorithm [7].

Various algorithms have been proposed for performing lifted inference for PLP [25,2], see [19] for a survey and comparison of the approaches.

In this paper we consider a simple PLP language (called *liftable PLP*) where programs contain clauses with a single annotated atom in the head with the same predicate. In this case, all the above approaches for lifted inference coincide and reduce to a computation similar to the one of the example above.

For this language, we discuss how to perform discriminative parameter learning (i.e., learning from positive and negative examples) by using EM or optimizing the likelihood with Limited-memory BFGS (LBFGS)[15].

A previous approach for performing lifted learning [8] targeted generative learning (no negative examples) for Markov Logic Networks, so it cannot be applied directly to PLP.

We also present LIFTCOVER for "LIFTed slipCOVER", an algorithm for performing discriminative structure learning of liftable PLP programs obtained from SLIPCOVER [3] by simplifying structure search and replacing parameter learning with one of the specialized approaches. We thus obtain LIFTCOVER-EM and LIFTCOVER-LBFGS that performs EM and LBFGS respectively.

We compare LIFTCOVER-EM, LIFTCOVER-LBFGS and SLIPCOVER on 12 dataset. The results show that LIFTCOVER-EM is nearly always faster and more accurate than SLIPCOVER while LIFTCOVER-LBFGS is often faster and similarly accurate than SLIPCOVER.

The paper is organized as follows: Section 2 introduces PLP under the distribution semantics, Section 3 presents the liftable PLP language, Sections 4 and 5 illustrate parameter and structure learning respectively, Section 6 discusses related work, Section 7 describes the experiments performed and Section 8 concludes the paper.

2 Probabilistic Logic Programming

We consider Probabilistic Logic Programming under the distribution semantics [20] for integrating logic programming with probability. PLP languages under the distribution semantics were shown expressive enough to represent a wide variety of domains [18,1]. A program in a language adopting the distribution semantics defines a probability distribution over normal logic programs called *instances* or *worlds*. Each normal program is assumed to have a total well-founded model [26]. Then, the distribution is extended to queries and the probability of a query is obtained by marginalizing the joint distribution of the query and the programs.

A PLP language under the distribution semantics with a general syntax is *Logic Programs with Annotated Disjunctions* (LPADs) [27]. We present here the semantics of LPADs for the case of no function symbols, if function symbols are

allowed see [17]. However, function symbols are usually not present the learning problems we consider, where the task is to learn a classifier rather than a program.

In LPADs, heads of clauses are disjunctions in which each atom is annotated with a probability. Let us consider an LPAD T with n clauses: $T = \{C_1, \ldots, C_n\}$. Each clause C_i takes the form: $h_{i1} : \Pi_{i1}; \ldots; h_{iv_i} : \Pi_{iv_i} :- b_{i1}, \ldots, b_{iu_i}$, where $h_{i1}, \ldots, h_{iv_i}$ are logical atoms, $b_{i1}, \ldots, b_{iu_i}$ are logical literals and $\Pi_{i1}, \ldots, \Pi_{iv_i}$ are real numbers in the interval $[0, 1]$ that sum to 1. $b_{i1}, \ldots, b_{iu_i}$ is indicated with $body(C_i)$. Note that if $v_i = 1$ the clause corresponds to a non-disjunctive clause. We also allow clauses where $\sum_{k=1}^{v_i} \Pi_{ik} < 1$: in this case the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{v_i} \Pi_{ik}$. We denote by $ground(T)$ the grounding of an LPAD T.

Each grounding $C_i\theta_j$ of a clause C_i corresponds to a random variable X_{ij} with values $\{1, \ldots, v_i\}$ where v_i is the number of head atoms of C_i. The random variables X_{ij} are independent of each other. An *atomic choice* [16] is a triple (C_i, θ_j, k) where $C_i \in T$, θ_j is a substitution that grounds C_i and $k \in \{1, \ldots, v_i\}$ identifies one of the head atoms. In practice (C_i, θ_j, k) corresponds to an assignment $X_{ij} = k$.

A *selection* σ is a set of atomic choices that, for each clause $C_i\theta_j$ in $ground(T)$, contains an atomic choice (C_i, θ_j, k). Let us indicate with S_T the set of all selections. A selection σ identifies a normal logic program l_σ defined as $l_\sigma = \{(h_{ik} \leftarrow body(C_i))\theta_j | (C_i, \theta_j, k) \in \sigma\}$. l_σ is called an *instance*, *possible world* or simply *world* of T. Since the random variables associated with ground clauses are independent, we can assign a probability to instances: $P(l_\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$.

We consider only *sound* LPADs where, for each selection σ in S_T, the well-founded model of the program l_σ chosen by σ is two-valued. We write $l_\sigma \models q$ to mean that the query q is true in the well-founded model of the program l_σ. Since the well-founded model of each world is two-valued, q can only be true or false in l_σ.

We denote the set of all instances by L_T. Let $P(L_T)$ be the distribution over instances. The probability of a query q given an instance l is $P(q|l) = 1$ if $l \models q$ and 0 otherwise. The probability of a query q is given by

$$P(q) = \sum_{l \in L_T} P(q, l) = \sum_{l \in L_T} P(q|l)P(l) = \sum_{l \in L_T : l \models q} P(l) \tag{1}$$

3 Liftable PLP

We restrict the language of LPADs by allowing only clauses of the form

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

in the program where all the clauses share the same predicate for the single atom in the head, let us call this predicate $target/a$ with a the arity. The literals in the body have predicates other than $target/a$ and are defined by facts and rules

that are certain, i.e., they have a single atom in the head with probability 1. Suppose there are n probabilistic clauses of the form above in the program. We call this language *liftable PLP*.

The problem is to compute the probability of a ground instantiation q of $target/a$. This can be done at the lifted level. We should first find the number of ground instantiations of clauses for $target/a$ such as the body is true and the head is equal to q. Suppose there are m_i such instantiations $\{\theta_{i1}, \ldots, \theta_{im_i}\}$, for rule C_i for $i = 1, \ldots, n$. Each instantiation θ_{ij} corresponds to a random variable X_{ij} taking values 1 with probability Π_i and 0 with probability $1 - \Pi_i$. The query q is true if at least one of the random variables for a rule takes value 1: $q = true \Leftrightarrow \bigvee_{i=1}^{n} \bigvee_{j=1}^{m_i} (X_{ij} = 1)$. In other words q is false only if no random variable takes value 1. All the random variables are mutually independent so the probability that none takes value 1 is $\prod_{i=1}^{n} \prod_{j=1}^{m_i} (1 - \Pi_i) = \prod_{i=1}^{n} (1 - \Pi_i)^{m_i}$ and the probability of q being true is $P(q) = 1 - \prod_{i=1}^{n} (1 - \Pi_i)^{m_i}$. So the probability of the query can be computed in logarithmic time once the number of clause instantiations with the body true: we just need to know the population sizes and do not need to perform knowledge compilation to perform inference.

We can picture the dependence of the random variable q from the variables of clause groundings as in Figure 1. Here the conditional probability table of q is that of an or: $P(q) = 1$ if at least one of its parents is equal to 1 and 0 otherwise. The variables from clause groundings are

$$\{X_{11}, \ldots, X_{1m_1}, X_{21}, \ldots, X_{2m_2}, \ldots, X_{n1}, \ldots, X_{nm_n}\}.$$

These are parentless variables, with X_{ij} having the CPT $P(X_{ij} = 1) = \Pi_i$ and $P(X_{ij} = 0) = 1 - \Pi_i$.

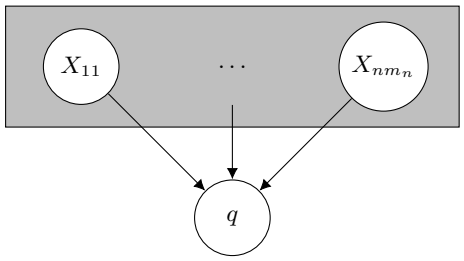

Fig. 1. Bayesian Network representing the dependency between the query q and the random variables associated with groundings of the clauses with the body true.

Example 1. Let us consider the UW-CSE domain [12] where the objective is to predict the "advised by" relation between students and professors. In this case the target predicate is $advisedby/2$ and a program for predicting such predicate may be

$advisedby(A, B) : 0.4 :-$
$\quad student(A), professor(B), publication(C, A), publication(C, B).$
$advisedby(A, B) : 0.5 :-$
$\quad student(A), professor(B), ta(C, A), taughtby(C, B).$

where $publication(A, B)$ means that A is a publication with author B, $ta(C, A)$ means that A is a teaching assistant for course C and $taughtby(C, B)$ means that course C is taught by B. The probability that a student is advised by a professor depends on the number of joint publications and the number of courses the professor teaches where the student is a TA, the higher these numbers the higher the probability.

Suppose we want to compute the probability of $q = advisedby(harry, ben)$ where *harry* is a student, *ben* is a professor, they have 4 joint publications and *ben* teaches 2 courses where *harry* is a TA. Then the first clause has 4 groundings with head q where the body is true, the second clause has 2 groundings with head q where the body is true and $P(advisedby(harry, ben)) = 1-(1-0.4)^4(1-0.5)^2 = 0.9676$.

4 Parameter Learning

Let us consider first the problem of learning the parameters of a liftable PLP T given a set of positive and negative examples. The problem can be expressed as follows: given a liftable PLP T, a set $E^+ = \{e_1, \ldots, e_Q\}$ of positive examples (ground atoms) and a set $E^- = \{e_{Q+1}, \ldots, e_R\}$ of negative examples (ground atoms) and a background knowledge B, find the parameters of T such that the likelihood

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

is maximized. The likelihood is given by the product of the probability of each example. The likelihood can be unfolded to

$$L = \prod_{q=1}^{Q} \left(1 - \prod_{l=1}^{n} (1 - p_l)^{m_{lq}}\right) \prod_{r=Q+1}^{R} \prod_{l=1}^{n} (1 - p_l)^{m_{lr}} \tag{2}$$

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 n is the number of clauses. We can aggregate the negative examples

$$L = \prod_{l=1}^{n} (1 - p_l)^{m_{l-}} \prod_{q=1}^{Q} \left(1 - \prod_{l=1}^{n} (1 - p_l)^{m_{lq}}\right) \tag{3}$$

where $m_{l-} = \sum_{r=Q+1}^{R} m_{lr}$.

We can maximize L using an Expectation Maximization (EM) algorithm since the X_{ij} variables are hidden. To perform EM, we need the following conditional probabilities (see the supplementary material [14] for the derivations):

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

This leads to Algorithm 1 (see the supplementary material [14] for a more detailed algorithm) that alternates between the Expectation, where the expectation of the counts are computed, and Maximization, where the parameters are updated.

Algorithm 1 Function EM

```
1: function EM(restarts, max_iter, ϵ, δ)
2:     BestLL ← −inf
3:     BestPar ← []
4:     for j ← 1, restarts do
5:         Π_i ← random for all i = 1, ..., n                ▷ n: number of rules
6:         LL = −inf
7:         iter ← 0
8:         repeat
9:             iter ← iter + 1
10:            LL_0 = LL                                     ▷ Start of expectation step
11:            c_i1 ← 0 for all i = 1, ... n                 ▷ c_ix: number of times X_ij takes value x for any j
12:            c_i0 ← m_i− for all i = 1, ..., n
13:            for q ← 1, Q do
14:                c_i1 += m_iq P(X_ij = 1|e_q) for all i = 1, ..., n
15:                c_i0 += m_iq P(X_ij = 0|e_q) for all i = 1, ..., n
16:            end for
17:            Compute LL                                    ▷ End of Expectation step
18:            Π_i = c_i1 / (c_i1 + c_i0) for all i = 1, ..., n   ▷ Maximization step
19:        until LL − LL_0 < ϵ ∨ LL − LL_0 < −LL · δ ∨ iter > max_iter
20:        if LL > BestLL then
21:            BestLL ← LL
22:            BestPar ← [Π_1, ..., Π_n]
23:        end if
24:    end for
25:    return BestLL, BestPar
26: end function
```

We can also optimize the likelihood using the gradient. The partial derivative of the likelihood with respect to p_i is (see the supplementary material [14] for the derivation):

$$\frac{\partial L}{\partial p_i}=\frac{L}{1-p_i}\left(\sum_{q=1}^{Q} m_{iq}\left(\frac{1}{P(e_q)}-1\right)-m_{i-}\right) \tag{4}$$

The equation $\frac{\partial L}{\partial p_i}=0$ does not admit a closed form solution, not even where there is a single clause, so we must use optimization to find the maximum of L.

We can use Limited-memory BFGS (LBFGS) [15], an optimization algorithm in the family of quasi-Newton methods that approximates the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm using a limited amount of

computer memory. LBFGS requires the computation of L and of its derivative at various points.

5 Structure Learning

The structure learning problem can be expressed as: 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, find a liftable PLP T such that the likelihood is maximized.

We solve this problem by first identifying good clauses guided by the log likelihood (LL) of the data. Clauses are found by a top-down beam search. The refinement operator adds a literal to the body of the clause C, the literal is taken from a bottom clause built as in Progol according to a user-defined language bias.

In each beam search iteration, the first clause of the beam is removed and all its refinements are computed. Each refinement Cl' is scored by performing parameter learning with $T = \{Cl'\}$ and using the resulting LL as the heuristic. The scored refinements are inserted back into the beam in order of heuristic. If the beam exceeds a maximum user-defined size, the last element is removed. Moreover, the refinements are added to a set of clauses CC.

Beam search is iterated a user-defined number of times or until the beam becomes empty. The output of this search phase is represented by the set CC of clauses. Then parameter learning is applied to the whole set CC, i.e., $T = CC$.

Algorithm 2 Function LIFTCOVER

```
1: function LIFTCOVER(NB, NI)
2:     Beam =InitialBeam                                   ▷ Bottom clauses building
3:     CC ← ∅
4:     Steps ← 1
5:     NewBeam ← []
6:     repeat
7:         Remove the first couple ((Cl, Literals), LL) from Beam      ▷ Remove the first clause
8:         Refs ←ClauseRefinements((Cl, Literals))  ▷ Find all refinements Refs of (Cl, Literals)
9:         for all (Cl', Literals') ∈ Refs do
10:            (LL'', {Cl''}) ←LearnWeights(I, {Cl'})
11:            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
12:            CC ← CC ∪ {Cl'}
13:        end for
14:        Beam ← NewBeam
15:        Steps ← Steps + 1
16:    until Steps > NI or Beam is empty
17:    (LL, Th) ←LearnWeights(I, CC, D, NEM, ϵ, δ)
18:    return Th
19: end function
```

Algorithm 2 shows the main LIFTCOVER function. Line 2 builds an initial beam $Beam$ consisting of bottom clauses. These clauses are generated using Progol's saturation method [13] starting from the language bias, and they will be refined by Function ClauseRefinements. For each mode declaration $modeh(r, s)$

in the language bias, an answer h for the goal $schema(s)$ is selected, where $schema(s)$ denotes the literal obtained from s by replacing all place-markers with distinct variables $X_1, \ldots, X_n$. The atom h is randomly sampled with replacement. Then a bottom clause is built starting from h. $Beam$ then will contain the clause with empty body $h : 0.5.$ for each bottom clause $h :- b_1, \ldots, b_m$. The clause is also associated with the set of literals $Literals = \{b_1, \ldots, b_m\}$ to be used during refinement.

6 Related Work

Lifted inference for PLP under the distribution semantics is surveyed in [19] that discuss three approaches.

LP^2 [2] uses an algorithm that extends First Order Variable Elimination for taking into account clauses that have variables in bodies not appearing in the head (existentially quantified variables). Weighted First Order Model Counting (WFOMC) [25] uses a Skolemization algorithm that eliminates existential quantifiers from a theory without changing its weighted model count. Kisynski and Poole [11] proposed an approach based on *aggregation parfactors* instead of parfactors that can represent noisy-OR models. These three approaches have X compared experimentally in [19] and WFOMC was found the fastest.

Relational Logistic Regression [9], a generalization of logistic regression to the relational case, can also be applied to PLP.

All these approaches reduce to the same algorithm for performing inference when the language is restricted to liftable PLP.

Lifted learning is still an open problem. An approach for performing lifted generative learning was proposed in [8]: while the paper discusses both weight and structure learning, it focuses on Markov Logic Networks and generative learning, so it is not directly applicable to the setting considered in this paper.

7 Experiments

LIFTCOVER[3] has been implemented in SWI-Prolog [28] using a porting[4] of YAP-LBFGS[5], a foreign language interface to libLBFGS[6].

LIFTCOVER has been tested on the following 12 real world datasets: the 4 classic benchmarks UW-CSE [12], Mutagenesis [23], Carcinogenesis [22], Mondial [21]; the 4 datasets Bupa, Nba, Pyrimidine, Triazine from `https://relational.fit.cvut.cz/`, and the 4 datasets Financial, Sisya, Sisyb and Yeast from [24][7]. We compare LIFTCOVER with SLIPCOVER because they have a very similar

[3] The code of the systems and the datasets are available at `https://bitbucket.org/machinelearningunife/liftcover`.

[4] `https://github.com/friguzzi/lbfgs`

[5] Developed by Bernd Gutmann.

[6] `http://www.chokkan.org/software/liblbfgs/`

[7] `https://dtai.cs.kuleuven.be/ACE/doc/`

structure learning algorithm, so the differences in performance should be due only to the different language and parameter learning approach. In particular, in all datasets except UW-CSE, the language bias for SLIPCOVER allows only one atom in the head, so the search space is very similar between LIFTCOVER and SLIPCOVER.

To evaluate the performance, we drew Precision-Recall curves and Receiver Operating Characteristics curves, computing the Area Under the Curve (AUC-PR and AUC-ROC respectively) with the methods reported in [4,6]. Statistics on all the domains are reported in Table 1. All experiments were performed on GNU/Linux machines with an Intel Xeon Haswell E5-2630 v3 (2.40GHz) CPU with 8GB of memory allocated to the job.

LIFTCOVER-EM was run with the following parameters for EM: $restarts = 1$, $max\_iter = 10$, $\epsilon = 10^{-4}$ and $\delta = 10^{-5}$. The default parameters have been used for libLBFGS. The parameters controlling structure learning have been set in the same way as in SLIPCOVER.

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

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.

Tables 2, 3 and 4 show the averages over the folds of AUC-ROC, AUC-PR and the execution time respectively of LIFTCOVER-EM, LIFTCOVER-LBFGS and SLIPCOVER.

LIFTCOVER-EM beats SLIPCOVER 7 times and ties twice in terms of AUC-ROC and beats SLIPCOVER 4 times and ties twice in terms of AUC-PR, with two cases (Sisya and Sisyb) where it is nearly as good. In terms of execution time, LIFTCOVER-EM is faster that SLIPCOVER in 9 cases and in the other three cases is nearly as fast. In 7 cases the gap is of one or more orders of magnitude (UW-CSE, Carcinognesis, Nba, Triazine, Sisya, Sisyb, Yeast).

AUC-ROC	LIFTCOVER-EM	LIFTCOVER-LBFGS	SLIPCOVER
UW-CSE	0.977	0.762	0.9681
Mutagen.	0.931	0.649	0.826
Mondial	0.663	0.643	0.630
Carcinogen.	0.766	0.472	0.695
Bupa	1.000	1.000	1.000
Nba	0.531	0.650	0.575
Pyrimidine	1.000	0.850	0.925
Triazine	0.713	0.760	0.544
Financial	0.432	0.535	0.568
Sisya	0.372	0.721	0.719
Sisyb	0.500	0.500	0.500
Yeast	0.786	0.721	0.733

Table 2. Results of the experiments in terms of average AUC-ROC.

AUC-PR	LIFTCOVER-EM	LIFTCOVER-LBFGS	SLIPCOVER
UW-CSE	0.220	0.263	0.286
Mutagen.	0.971	0.725	0.920
Mondial	0.763	0.723	0.776
Carcinogen.	0.672	0.561	0.745
Bupa	1.000	1.000	1.000
Nba	0.550	0.705	0.550
Pyrimidine	1.000	0.819	0.956
Triazine	0.734	0.760	0.560
Financial	0.126	0.187	0.173
Sisya	0.706	0.706	0.708
Sisyb	0.286	0.286	0.287
Yeast	0.502	0.448	0.428

Table 3. Results of the experiments in terms of average AUC-PR.

Time	LIFTCOVER-EM	LIFTCOVER-LBFGS	SLIPCOVER
UW-CSE	8.054	178.6	103.4
Mutagen.	12.77	122.8	12.11
Mondial	5.911	3.984	6.490
Carcinogen.	7.850	76.49	25568
Bupa	0.243	1.239	1.349
Nba	0.599	1.036	386.0
Pyrimidine	54.99	126.1	54.62
Triazine	56.69	109.1	728.2
Financial	0.235	0.246	0.178
Sisya	0.932	2.252	45.75
Sisyb	0.226	0.412	37.00
Yeast	0.502	69.30	202.4

Table 4. Results of the experiments in terms of average time.

LIFTCOVER-LBFGS beats SLIPCOVER 4 times (in Sisya they are very close) and ties twice in terms of AUC-ROC and beats SLIPCOVER 4 times and ties once in terms of AUC-PR, with two cases where it is nearly as good. In terms of execution time, LIFTCOVER-LBFGS beats SLIPCOVER 8 times, with one case where SLIPCOVER is nearly as fast. In 4 cases the gap is of one or more orders of magnitude (Carcinogenesis, Nba, Sisya, Sisyb). In Mutagenesis and Pyrimidine LIFTCOVER-LBFGS takes about twice as much as SLIPCOVER.

Overall we see that both lifted algorithms are usually faster, sometimes by a large margin, with respect to SLIPCOVER, especially LIFTCOVER-EM. Surprisingly, this system often finds better quality solutions, even if the language is more restricted, probably due to reduced overfitting,

Between the two lifted algorithms, the EM version wins 6 times and ties twice with respect to AUC-ROC and wins 5 times and ties 3 times with respect to AUC-PR. In terms of execution times, the EM version wins on all dataset except Mondial, with differences below one order of magnitude except UW-CSE and Yeast. So LIFTCOVER-EM appears to be superior both in terms of solution quality and of computation time. EM seems to be better at escaping local maxima and cheaper than LBFGS, possibly also due to the fact that LBFGS may require a careful tuning of parameters and that it is implemented as a foreign language library.

8 Conclusions

We have presented an algorithm that learns the structure of liftable probabilistic logic programs using either EM or LBFGS for parameter learning. The results show that LIFTCOVER-EM is faster than SLIPCOVER and often more accurate, while LIFTCOVER-LBFGS is slightly slower and slightly less accurate. In the future we plan to test the influence of settings in LBFGS and to add regularization to parameter learning in order to avoid overfitting.

References

1. Alberti, M., Bellodi, E., Cota, G., Riguzzi, F., Zese, R.: `cplint` on SWISH: Probabilistic logical inference with a web browser. Intell. Artif. 11(1) (2017)
2. Bellodi, E., Lamma, E., Riguzzi, F., Costa, V.S., Zese, R.: Lifted variable elimination for probabilistic logic programming. Theor. Pract. Log. Prog. 14(4-5), 681–695 (2014)
3. Bellodi, E., Riguzzi, F.: Structure learning of probabilistic logic programs by searching the clause space. Theor. Pract. Log. Prog. 15(2), 169–212 (2015)
4. Davis, J., Goadrich, M.: The relationship between Precision-Recall and ROC curves. In: ECML 2006. pp. 233–240. ACM (2006)
5. De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Mach. Learn. 100(1), 5–47 (2015)
6. Fawcett, T.: An introduction to ROC analysis. Pattern Recognit. Lett. 27, 861–874 (2006)

7. Gordon, D.M.: A survey of fast exponentiation methods. J. Algorithms 27(1), 129 – 146 (1998)
8. Haaren, J.V., den Broeck, G.V., Meert, W., Davis, J.: Lifted generative learning of markov logic networks. Mach. Learn. 103(1), 27–55 (2016)
9. Kazemi, S.M., Buchman, D., Kersting, K., Natarajan, S., Poole, D.: Relational logistic regression. In: Baral, C., Giacomo, G.D., Eiter, T. (eds.) KR 2014. AAAI Press (2014)
10. Kimmig, A., Mihalkova, L., Getoor, L.: Lifted graphical models: a survey. Mach. Learn. 99(1), 1–45 (2015)
11. Kisynski, J., Poole, D.: Lifted aggregation in directed first-order probabilistic models. In: Boutilier, C. (ed.) IJCAI 2009. pp. 1922–1929 (2009)
12. Kok, S., Domingos, P.: Learning the structure of Markov Logic Networks. In: ICML 2005. pp. 441–448. ACM (2005)
13. Muggleton, S.: Inverse entailment and Progol. New Generat. Comput. 13, 245–286 (1995)
14. Nguembang Fadja, A., Riguzzi, F.: Supplementary material, `http://ds.ing.unife.it/~friguzzi/Papers/ILP17-supp.pdf`
15. Nocedal, J.: Updating quasi-newton matrices with limited storage. Math. Comput. 35(151), 773–782 (1980)
16. Poole, D.: The Independent Choice Logic for modelling multiple agents under uncertainty. Artif. Intell. 94, 7–56 (1997)
17. Riguzzi, F.: The distribution semantics for normal programs with function symbols. Int. J. Approx. Reason. 77, 1 – 19 (October 2016)
18. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., Cota, G.: Probabilistic logic programming on the web. Softw.-Pract. Exper. 46(10), 1381–1396 (October 2016)
19. 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 (January 2017)
20. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) ICLP 1995. pp. 715–729. MIT Press, Cambridge, Massachusetts (1995)
21. Schulte, O., Khosravi, H.: Learning graphical models for relational data via lattice search. Mach. Learn. 88(3), 331–368 (2012)
22. Srinivasan, A., King, R.D., Muggleton, S., Sternberg, M.J.E.: Carcinogenesis predictions using ILP. In: Lavrac, N., Dzeroski, S. (eds.) ILP 1997. LNCS, vol. 1297, pp. 273–287. Springer Berlin Heidelberg (1997)
23. 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)
24. Struyf, J., Davis, J., Page, D.: An efficient approximation to lookahead in relational learners. In: ECML 2006. pp. 775–782. LNCS, Springer (2006)
25. Van den Broeck, G., Meert, W., Darwiche, A.: Skolemization for weighted first-order model counting. In: Baral, C., Giacomo, G.D., Eiter, T. (eds.) KR 2014. pp. 111–120. AAAI Press (2014)
26. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM 38(3), 620–650 (1991)
27. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic Programs With Annotated Disjunctions. In: ICLP 2004. LNCS, vol. 3132, pp. 431–445. Springer (2004)
28. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theor. Pract. Log. Prog. 12(1-2), 67–96 (2012)