Closed-Form Solutions in Learning Probabilistic Logic Programs by Exact Score Maximization

Francisco Henrique Otte Vieira de Faria[1],
Fabio Gagliardi Cozman[1], and Denis Deratani Mauá[2]

[1] Escola Politécnica, Universidade de São Paulo
[2] Instituto de Matemática e Estatística, Universidade de São Paulo

Abstract. We present an algorithm that learns acyclic propositional probabilistic logic programs from complete data, by adapting techniques from Bayesian network learning. Specifically, we focus on score-based learning and on exact maximum likelihood computations. Our main contribution is to show that by restricting any rule body to contain at most two literals, most needed optimization steps can be solved exactly. We describe experiments indicating that our techniques do produce accurate models from data with reduced numbers of parameters.

Keywords: Probabilistic logic programming, Score-based structure learning of Bayesian networks.

1 Introduction

The goal of this paper is to present techniques that learn probabilistic logic programs (PLPs) from complete data. Probabilistic logic programs have been explored for some time [10, 13, 14, 16, 20], and are now the object of significant literature [18, 19]; yet there is much to be developed when it comes to rule learning. In this paper we wish to examine the extent to which, for some classes of PLPs, we can find the exact optimal PLP with respect to some score. That is, we look for classes of PLPs have some guarantees concerning the optimization of scores given data.

With this broad goal in mind, here we focus on algorithms that learn, in the sense of maximization of minimum description length, acyclic sets of propositional rules with at most two literals in their bodies. This is admitedly a restricted class of PLPs, but note that these PLPs can encode all "noisy" Boolean circuits and can serve as starting point for more ambitious investigations.

Our main contribution is to show that, by focusing on this restricted class of PLPs, we can *exactly*, in *constant time*, solve most optimizations involved in description length minimization.

Because acyclic propositional PLPs are intimately related to Bayesian networks [17], we employ insights from Bayesian network learning, in particular resorting to score-based learning where the score is based on minimum description length [11]. However most of our arguments apply to any decomposable

score that depends on likelihood; for example, analogues of K2 and BDeu scores [11] could be easily adopted.

We briefly review the main features of relevant PLPs in Section 2. We then introduce the score maximization problem we face (Section 3); this follows by direct adaptation of methods from Bayesian network learning. In Sections 4 and 5 we derive an algorithm for score maximization that relies both on exact maximization of polynomial equations for local optimization and on constraint programming for global optimization. Experiments and results indicating that our method is successful in recovering accurate models are reported in Section 6.

2 Probabilistic logic programs: a (very) brief review

Take a fixed vocabulary consisting of logical variables $X, X_1, \ldots$, predicates $\mathsf{r}, \mathsf{r}_r, \ldots$, and constants $a, b, \ldots$. A *term* is a constant or a logical variable; an *atom* is written as $\mathsf{r}(t_1, \ldots, t_n)$, where r is a predicate of arity n and each t_i is a term (a 0-arity atom is written as r). An atom is *ground* if it does not contain logical variables. A *normal logic program* consists of rules

$$A_0 :- A_1, \ldots, A_m, \textbf{not } A_{m+1}, \ldots, \textbf{not } A_n\textbf{.},$$

where the A_i are atoms. The *head* of this rule is A_0; the right-hand side is the *body*. A rule without a body, written $A_0.$, is a *fact*. A *literal* is either A (positive) or **not** A (negative), where A is an atom.

In this paper we only consider propositional programs; that is, programs without logical variables.

The *Herbrand base* is the set of all ground atoms built from constants and predicates in a program. We do not consider functions in this paper, hence every Herbrand base is finite.

The *dependency graph* of a program is a directed graph where each predicate is a node, and where there is an edge from a node B to a node A if there is a rule where A appears in the head and B appears in the body (if B appears right after **not**, the edge is *negative*; otherwise, it is *positive*). The *grounded* dependency graph is the dependency graph of the propositional program obtained by grounding.

An acyclic program is one with an acyclic grounded dependence graph. In this paper we only consider acyclic programs.

There are several ways to combine logic programming and probabilities; in this work we adopt a popular combination associated with Sato's distribution semantics [16, 20]. A *probabilistic logic program*, abbreviated PLP, is a pair $\langle \mathbf{P}, \mathbf{PF} \rangle$ consisting of a normal logic program $\mathbf{P}$ and a set of *probabilistic facts* $\mathbf{PF}$. A probabilistic fact is a pair consisting of an atom A and a probability value α, written as $\alpha :: A\textbf{.}$ (here we adopt the syntax of the ProbLog package [8]). Note that we allow a probabilistic fact to contain logical variables.

To build the semantics of a PLP, we first take its grounding. Suppose we have a PLP with n grounded probabilistic facts (some of them may be given as propositional probabilistic facts, others are obtained by grounding non-propositional

probabilistic facts). There are then 2^n ways to select subsets of these propositional probabilistic facts. For each such subset, we can construct the normal logic program consisting of the non-probabilistic part of the PLP plus the atoms in the selected probabilistic facts. That is, for each probabilistic fact $\alpha :: A.$, either keep fact $A.$ with probability α, or erase $A.$ with probability $1 - \alpha$. A *total choice* is simply a subset of the set of ground probabilistic facts that is selected to be kept (other grounded probabilistic facts are discarded). So, for any total choice θ we obtain a normal logic program, denoted by $\mathbf{P} \cup \mathbf{PF}^{\downarrow\theta}$, with probability $\prod_{A_i \in \theta} \alpha_i \prod_{A_i \notin \theta} (1 - \alpha_i)$. Hence the distribution over total choices induces a distribution over logic programs.

Note that if $\mathbf{P}$ is acyclic, then for any total choice we have that $\mathbf{P} \cup \mathbf{PF}^{\downarrow\theta}$ is acyclic. So the semantics of the whole PLP is relatively simple to describe: the probability distribution over total choices induces a probability distribution over interpretations, such that for each fixed total choice we obtain the truth assignment of all atoms by applying the rules in appropriate order (that is, if all atoms in the body of a rule are true, the head is true, and this is the only way to render a head true).

A common pattern in PLPs is a pair rule/probabilistic fact such as

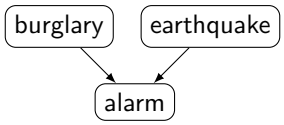
$$A_0 :- A_1, \ldots, A_m, \mathbf{not}\ A_{m+1}, \ldots, \mathbf{not}\ A_n, F., \quad \alpha :: F.,$$

meaning that with probability $1 - \alpha$ the whole rule is not "activated". We write such a construct as

$$\alpha :: A_0 :- A_1, \ldots, A_m, \mathbf{not}\ A_{m+1}, \ldots, \mathbf{not}\ A_n.,$$

again adopting the syntax of ProbLog [8].

Example 1 *Here is an acyclic propositional* PLP*:*



0.1 :: burglary. 0.2 :: earthquake.
0.9 :: alarm :− burglary, earthquake.
0.8 :: alarm :− burglary, **not** earthquake.
0.7 :: alarm :− **not** burglary, earthquake.
0.1 :: alarm :− **not** burglary, **not** earthquake.

The dependency graph is presented at the right. □

The class of logic programs where each rule has at most k atoms in the body is denoted LP(k) [5]; we analogously write PLP(k) to denote the class of PLPs where each rule has at most k literals in the body. And we use AP-PLP to refer to acyclic propositional PLPs. In this paper we focus on the class AP-PLP(2).

3 Learning by score maximization

One may wish to learn, from data, both the rules of a PLP and the associated probabilities; that is, both the *structure* and the *parameters* of the PLP. A general strategy in structure learning is to add probabilistic estimation to Inductive

Logic Programming; usually such a strategy is referred to as Probabilistic Inductive Logic Programming [18, 19]. Typically such mix of probabilities and logic requires a search over the space of rules, under the assumption that some examples are "positive" and must receive high probability, while other examples are "negative" and must receive low probability. Search schemes vary and are almost universally based on heuristic measures, to guarantee that large datasets can be processed [1, 6, 7, 25].

Another general strategy, when learning a probabilistic model, is to maximize a score that quantifies the fit between model and data. This is the strategy most often employed to learn the the structure of Bayesian networks, and the strategy adopted in this paper. To grasp the main ideas behind score-based structure learning, we first review the interplay between Bayesian networks and probabilistic logic programs.

3.1 Bayesian networks and probabilistic logic programs

As noted in the Introduction, acyclic propositional PLPs are closely related to Bayesian networks [17]. Recall that a Bayesian network consists of a directed acyclic graph where each node is a random variable, and a probability distribution over the same random variables, such that the distribution satisfies the following Markov condition: a variable X is independent of its nondescendants nonparents given its parents [15]. For any Bayesian network, its directed acyclic graph is referred to as its "structure". The parents of a variable X, denoted by $\text{PA}[X]$, are the nodes/variables that point to X. In this paper every random variable has finitely many values (indeed all of them are binary). When a conditional probability distribution over random variables with finitely many values is encoded using a table, the latter table is often referred to as a *CPT*.

Any Bayesian network over binary variables can be encoded by an acyclic propositional PLP; conversely, any acyclic propositional PLP can be viewed as a Bayesian network. The last statement should be clear from Example 1: the Bayesian network described by the PLP has the structure given by the dependency graph, and the parameters of the network are just the probabilities associated with probabilistic facts and rules. The converse is equally simple to show, and consists of easily translating the probability assignments into rules and probabilistic facts, as argued by Poole [16, 17].

The "structure" of an acyclic PLP is related to the "structure" of the Bayesian network associated to the PLP. In fact, the dependency graph of the grounded PLP is the structure of the corresponding Bayesian network. However, a PLP can specify significantly more detail about the underlying probability distributions. Suppose, for instance, that the distribution of a binary variable X, with parents Y and Z, is given by a NoisyOr gate [15]; that is, $X = YY' + ZZ' - YY'ZZ'$, with $\mathbb{P}(Y' = 1) = \alpha$ and $\mathbb{P}(Z' = 1) = \beta$. In this case the conditional probability distribution of X given (Y, Z) is fully specified by two numbers (α and β), instead of the four numbers that a complete specification requires. Note that a small set of probabilistic facts and rules would have no trouble in encoding exactly this NoisyOr gate with the needed two parameters. This is attractive: if a distribution

can be at all captured by a small number of parameters, a PLP may be able to do so.

Of course, there are other ways to capture conditional distributions with "local" structure; that is, distributions that require few parameters to yield the probabilities for some variable given its parents. One notable example in the literature is the use of trees to represent conditional distributions [9]. The representation of a conditional probability distribution using trees is sometimes referred to as a CPT-tree [2]. Now it should be clear that CPT-trees and probabilistic rules do not have the same expressivity; for instance a CPT-tree requires three parameters to specify the NoisyOr gate in the previous paragraph (assuming a convention that leaves unspecified branches as zero), while rules can specify the NoisyOr gate with two parameters. So the question as to whether representations based on probabilistic rules are more compact than other representation is meaningful, and this is the sort of abstract question we wish to address with the current paper.

3.2 Score-based structure learning of Bayesian networks

Several successful structure learning methods are based on score maximization, where a score $s(B, D)$ gets a Bayesian network structure B and a dataset D, and yields a number that indicates the fit between both. We assume that D is complete (that is, there is no missing data) and consists of N observations of all random variables of interest. Sensible scores balance the desire to maximize likelihood against the need to constrain the number of learned parameters. It is well-known that if the one maximizes only the likelihood, then the densest networks are always obtained [11]. One particularly popular score is based on minimum description length guidelines; the score is:

$$s_{\mathsf{MDL}}(B, D) = \mathsf{LL}_D(B) - \frac{|B| \log N}{2}, \tag{1}$$

where: $|B|$ is the number of parameters needed to specify the network, and $\mathsf{LL}_D(B)$ is the log-likelihood at the maximum likelihood estimates (that is, the logartithm of $p(D, \Theta^{B,D}|B)$ with p denoting the probability density of observations given a Bayesian network structure B for probability values $\Theta^{B,D}$. The latter values are obtained again by likelihood maximization; that is, $\Theta^{B,D} = \arg\max_\Theta p(\Theta, D|B)$. We adopt the s_{MDL} score throughout this paper.

The MDL score, as other popular scores such as the K2 and BDeu scores, is *decomposable*; that is, the score is a sum of *local scores*, each one a function of a variable and its parents. We call *family* a set consisting of a variable and its parents.

The current technology on structure learning of Bayesian networks can handle relatively large sets of random variables [4, 22, 3]. Most existing methods proceed in two steps: first calculate the local score for every possible family; then maximize the global score, usually either by integer programming [4] or by constraint programming [22]. When one deals with structure learning of

Bayesian networks where conditional probability distributions are encoded by CPTs, then maximum likelihood estimates $\Theta^{B,D}$ are obtained in closed form: they are, in fact, simply relative frequencies. If we denote by θ_{ijk} the probability $\mathbb{P}(X_i = x_{ij}|\text{PA}[X_i] = \pi_k)$, where π_k is the kth configuration of the parents of X_i, then the maximum likelihood estimate is N_{ijk}/N_{ij}, where N_{ijk} is the number of times the configuration $\{X_i = x_{ij}, \text{PA}[X_i] = \pi_k\}$ occurs in D, and N_{ik} is the number of times the configuration $\{\text{PA}[X_i] = \pi_k\}$ occurs in D. Thus the calculation of the scores is not really taxing; the real computational effort is spent running the global optimization step.

4 A score-based learning algorithm for the class AP-PLP(2)

Our goal is to learn a PLP that maximizes the MDL score with respect to the complete dataset D, with the restriction that resulting PLPs must belong to the class AP-PLP(2). We do so by following the two-step scheme discussed in the previous section: first, we must compute the local score for each possible family, and then we must run a global maximization step to obtain the whole PLP.

We must of course translate the language of acyclic propositional PLPs into the language of Bayesian networks. Each proposition in our vocabulary is viewed as a binary random variable (0 is false and 1 is true), and the propositions that appear in the body of a rule are the parents of the proposition in the head of the rule. This is clearly consistent with the correspondence between PLPs and Bayesian networks. Our dataset D is therefore a collection of N observations of the propositions/variables of interest.

Because the MDL score is decomposable, we can globally maximize it by first maximizing each local score separately, and then running a global maximization step that selects a family for each variable. But there is a difference between usual Bayesian network learning and PLP learning: even within a family, we must choose the rule set that relates the head of the family with its parents.

Example 2 *Suppose we have propositions $A_1, A_2, \ldots, A_n$. We must contemplate every possible family; for instance, A_1 may be a root (no parents), or A_1 may be the child of A_2, or the child of A_3, or the child of A_2 and A_3, and so on (we are restricted to at most two parents). Suppose we focus on the family $\{A_1, A_2, A_3\}$ for proposition/variable A_1; that is, A_1 is the head and $\{A_2, A_3\}$ appear in the bodies. How many possible rule sets can we build? Indeed, many. For instance, we may have a simple rule such as*

$$\theta :: A_1 :- A_2, A_3 \bullet.$$

This single rule is equivalent to the following CPT:

A_2	A_3	$\mathbb{P}(A_1 = \text{true}\|A_2, A_3)$
false	false	0
false	true	0
true	false	0
true	true	θ

Or we may have three rules such as

$$\theta_1 :: A_1. \qquad \theta_2 :: A_1 :- A_2, \textbf{not } A_3. \qquad \theta_3 :: A_1 :- \textbf{not } A_2, \textbf{not } A_3. \tag{2}$$

These three rules are equivalent to the following CPT:

A_2	A_3	$\mathbb{P}(A_1 = \mathsf{true} \mid A_2, A_3)$
false	false	$\theta_1 + \theta_3 - \theta_1\theta_3$
false	true	θ_1
true	false	$\theta_1 + \theta_2 - \theta_1\theta_2$
true	true	θ_1

Two lines of this table contain nonlinear functions of the parameters, a fact that complicates the likelihood maximization step. □

How many different rule sets we must consider? Suppose first that we have a family containing only the head A. Then there is a single rule, the probabilistic fact $\theta :: A.$. If we instead have a family containing the head A and the body proposition B, there are six other options to consider:

$\theta :: A :- B.$	$\theta :: A :- \textbf{not } B.$	$\theta_1 :: A :- B.$ $\theta_2 :: A.$
$\theta_1 :: A :- \textbf{not } B.$ $\theta_2 :: A.$	$\theta_1 :: A :- B.$ $\theta_2 :: A :- \textbf{not } B.$	$\theta_1 :: A :- B.$ $\theta_2 :: A :- \textbf{not } B.$ $\theta_3 :: A.$

Now suppose we have a head A with parents B and C. First, there are 9 possible rules where A is the head and no proposition other than B or C appears in the body.[3] Each one of the 2^9 subsets of these rules is a possible rule set for this family; however, 13 of these subsets do not mention either B or C. Thus there are $2^9 - 13 = 499$ new rule sets to evaluate.

In any case, for each rule set we consider, we must maximize a likelihood function that may be nonlinear on the parameters. For instance, take the set of three rules in Expression (2). Denote by N_{000} the number of times that $\{A_1 = \mathsf{false}, A_2 = \mathsf{false}, A_3 = \mathsf{false}\}$ appear in the dataset; by N_{001} the number of times that $\{A_1 = \mathsf{false}, A_2 = \mathsf{false}, A_3 = \mathsf{true}\}$ appear in the dataset, and likewise each N_{ijk} stands for the number of times a particular configuration of A_1, A_2 and A_3 appear in the dataset. Then the local likelihood that must be maximized for this candidate family is

$$(\theta_{1;3})^{N_{100}}(1-\theta_{1;3})^{N_{000}}(\theta_{1;2})^{N_{110}}(1-\theta_{1;2})^{N_{010}}\theta_1^{N_{101}+N_{111}}(1-\theta_1)^{N_{001}+N_{011}},$$

where we use, here and in the remainder of the paper, $\theta_{i;j}$ to denote $\theta_i + \theta_j - \theta_i\theta_j$.

[3] These 9 rules are: $\theta :: A.$, $\theta :: A :- B.$, $\theta :: A :- \textbf{not } B.$, $\theta :: A :- C.$, $\theta :: A :- \textbf{not } C.$, $\theta :: A :- B, C.$, $\theta :: A :- B, \textbf{not } C.$, $\theta :: A :- \textbf{not } B, C.$, $\theta :: A :- \textbf{not } B, \textbf{not } C.$.

Hence, even at the local level, we face a non-trivial optimization problem. We discuss the solution of this problem in the next section. For now we assume that the problem has been solved; that is, each family is associated with a local score (the log-likelihood of that family, with parameters that maximize likelihood, minus a penalty on the number of parameters). Once the local score are ready, we resort to the constraint-programming algorithm (CPBayes) by Van Beek and Hoffmann [22] to run the global optimization step, thus selecting families so as to have an acyclic PLP. Clearly a selection of families leads to a PLP, as each family is associated with the rule set that maximizes the local score.

The CPBayes algorithm defines a set of constraints that must be satisfied in the Bayesian network learning problem, and seeks for an optimal solution based on a depth-first BnB search. When trying to expand a node in the search tree, two conditions are verified: (1) whether constraints are satisfied, and (2) whether a lower bound estimate of the cost does not exceed the current upper bound. The constraint model includes dominance constraints, symmetry-breaking constraints, cost-based pruning rules, and a global acyclicity constraint. We remark that other approaches for the global optimization can be used, and our contribution is certainly not due to our use of CPBayes in the global optimization step. Thus we do not dwell on this second step.

5 Computing the local score

In this section we address the main novel challenge posed by score-based learning of PLPs; namely, the computation of local scores. As discussed in the previous section, this is not a trivial problem for two reasons. First, there are too many sets of rules to consider. Second, likelihood maximization for each rule set may have to deal with nonlinear expressions for probability values.

We deal with the first problem by pruning rule sets; that is, by developing techniques that allow us to quickly eliminate many candidate rule sets.

First of all, we can easily discard rule sets that assign zero probability to some configuration observed in the dataset (for instance, the first rule in Example 2 can be discarded if we observe $\{A_2 = \mathsf{false}, A_3 = \mathsf{false}\}$).

Second, and more importantly, suppose that we are learning rules with at most k literals in the body. With 2^k rules we can have a rule for each configuration of the parents: Example 1 illustrates this scenario. Note that for such "disjoint" rules, likelihood maximization is simple as it is the same as for usual CPT. And because *any* CPT can be exactly built with such 2^k rules, any set of rules with more than 2^k rules cannot have higher likelihood, and thus cannot be optimal (as the penalty for the number of parameters increases). In fact, any other rule set with 2^k rules that are not disjoint can be also discarded; these sets can only produce the same likelihood, and will pay the same penalty on parameters, but they will be more complex to handle. Thus we must only deal with rule sets with at most $2^k - 1$ rules, plus the one set of 2^k "disjoint" rules. Hence:

Table 1. A probability pattern shared by several rule sets; first column presents the rule sets, and following columns display the probability values for the configurations of A_1, A_2, A_3.

Rule sets	0,0,0	0,0,1	0,1,0	0,1,1	1,0,0	1,0,1	1,1,0	1,1,1
$\theta_1 :: A_1 :- A_2.$ $\theta_2 :: A_1 :- A_2, A_3.$ $\theta_3 :: A_1 :- A_2, \mathbf{not}\ A_3.$	1	1	$1-\theta_{1;3}$	$1-\theta_{1;2}$	0	0	$\theta_{1;3}$	$\theta_{1;2}$
$\theta_1 :: A_1 :- A_2.$ $\theta_2 :: A_1 :- A_2, A_3.$	1	1	$1-\theta_1$	$1-\theta_{1;2}$	0	0	θ_1	$\theta_{1;2}$
$\theta_1 :: A_1 :- A_2.$ $\theta_3 :: A_1 :- A_2, \mathbf{not}\ A_3.$	1	1	$1-\theta_{1;2}$	$1-\theta_1$	0	0	$\theta_{1;2}$	θ_1
$\theta_2 :: A_1 :- A_2, A_3.$ $\theta_3 :: A_1 :- A_2, \mathbf{not}\ A_3.$	1	1	$1-\theta_2$	$1-\theta_1$	0	0	θ_2	θ_1

- If we have a family with $k = 1$, we only need to look at sets of one rule plus one set containing two disjoint rules; that is, we only have to consider:

$$\theta :: A :- B. \quad \Big| \quad \theta :: A :- \mathbf{not}\ B. \quad \Big| \quad \begin{array}{c} \theta_1 :: A :- B. \\ \theta_2 :: A :- \mathbf{not}\ B. \end{array}$$

  Note that the last of these three rule sets corresponds to a typical CPT, while the first two rule sets genuinely reduce the number of parameters in the model. Estimates that maximize likelihood are easily computed in all three cases.
- Now if we have a family with $k = 2$, we only need to look at sets of up to three rules, plus one set containing four disjoint rules. There are 4 sets consisting each of one rule, 30 sets consisting of two rules each, and 82 sets consisting of three rules each (we cannot list them all here due to space constraints). In this case probability values may have nonlinear expressions as discussed in connection with Expression (2). Thus we still have a challenging optimization problem, where we must find a rule set out of many.

To address the difficulty mentioned in the previous sentence, we resort to a third insight: many of the rule sets obtained for $k = 2$ are actually restricted versions of a few patterns. As an example, consider Table 1. There we find four different rule sets, some with two rules, and one with three rules. The form of their likelihoods is the same, sans some renaming of parameters. Note that the maximum likelihood of the first three rule sets can always be attained by the likelihood of the last rule set; consequently, it makes sense only to retain the last pattern, which consists of disjoint rules.

By doing this additional pruning, we reach 14 distinct rule sets; amongst them we must find a rule set that maximizes likelihood. The remaining difficulty is that probability values are nonlinear functions of parameters, as we have already indicated. However, most of them lead to likelihood expressions that can be *exactly* maximized. For instance, consider the following expression, a likelihood pattern that several rule sets (consisting of two rules) produce:

$$\theta_1^{N_0}(1-\theta_1)^{N_1}(\theta_1+\theta_2-\theta_1\theta_2)^{M_0}(1-(\theta_1+\theta_2-\theta_1\theta_2))^{M_1};$$

this function is maximized by:

$$\theta_1 = \frac{N_0}{N_0 + N_1}, \qquad \theta_2 = \frac{N_1 M_0 - N_0 M_1}{N_1 M_0 + N_1 M_1}.$$

Similarly, consider the following likelihood pattern (produced by rules sets consisting of three rules):

$$\theta_1^{N_0}(1-\theta_1)^{N_1}(\theta_1+\theta_2-\theta_1\theta_2)^{M_0}(1-(\theta_1+\theta_2-\theta_1\theta_2))^{M_1}$$
$$(\theta_1+\theta_3-\theta_1\theta_2)^{Q_0}(1-(\theta_1+\theta_3-\theta_1\theta_3))^{Q_1};$$

this function is maximized by:

$$\theta_1 = \frac{N_0}{N_0 + N_1}, \qquad \theta_2 = \frac{N_1 M_0 - N_0 M_1}{N_1 M_0 + N_1 M_1}, \qquad \theta_3 = \frac{N_1 Q_0 - N_0 Q_1}{N_1 Q_0 + N_1 Q_1}.$$

Due to space constraints we omit the complete list of likelihood patterns and their maximizing parameters. There are only four patterns that do not seem to admit closed form solutions. Here is one example:

$$\theta_1^{N_0}(1-\theta_1)^{N_1}(\theta_1+\theta_2-\theta_1\theta_2)^{M_0}(1-(\theta_1+\theta_2-\theta_1\theta_2))^{M_1}$$
$$(\theta_1+\theta_3-\theta_1\theta_3)^{Q_0}(1-(\theta_1+\theta_3-\theta_1\theta_3))^{Q_1}(\theta_1+\theta_2+\theta_3-\theta_1\theta_2-\theta_1\theta_3-\theta_2\theta_3+\theta_1\theta_2\theta_3)^{R_0}$$
$$(1-(\theta_1+\theta_2+\theta_3-\theta_1\theta_2-\theta_1\theta_3-\theta_2\theta_3+\theta_1\theta_2\theta_3))^{R_1}.$$

By taking logarithms and derivatives, these maximization problems can then turned into the solution of systems of polynomial equations. Such systems can be solved exactly but slowly, or approximately by very fast algorithms (as we comment in the next section).

The procedure we have developed is summarized by Algorithm 1. We firstly generate all possible combinations of rules for all possible families, a possible family consisting of a variable and its parent candidates. Combinations with ensured lower score or zero likelihood are then pruned and parameters are locally optimized for each of the combinations left. Each family is then associated with the combination of rules that gives it the highest score. Finally, a global score maximization algorithm is used to select the best family candidates.

6 Experiments

To validate our methods, we have implemented the learning algorithm described previously, and tested it with a number of datasets. Our goal was to examine whether the algorithm actually produces sensible PLPs with less parameters than corresponding Bayesian networks based on explicit CPTs.

The algorithm was implemented in Python and experiments were performed on a Unix Machine with Intel core i5 (2.7 GHz) processor and 8 GB 1867 MHz DDR3 SDRAM. For local optimization of the likelihood scores, in the few cases where that was needed, we used, and compared, two different algorithms: (1)

Algorithm 1 Learning algorithm for class AP-PLP(2).

```
collect variables from dataset
for each family of variables in dataset do
    build all possible rules
    build all possible combinations of rules
    gather rule sets into patterns
    for each pattern do
        prune combinations with ensured lower score
        prune combinations with zero likelihood
        for each combination left do
            if there is an exact solution to the likelihood maximization problem then
                calculate parameters
            else
                run numeric (exact or approximate) likelihood maximization
        calculate local scores
    for each family do
        associate best rule set with family
call CPBayes algorithm to maximize global score
```

Limited-memory BFGS (L-BFGS) and (2) the Basin-hopping algorithm [23]. Both methods are implemented in the Python library scipy.optimize. The L-BFGS algorithm approximates the BroydenFletcherGoldfarbShanno (BFGS) algorithm [24], which is an iterative method for solving unconstrained nonlinear optimization problems. The L-BFGS algorithm represents with a few vectors an approximation to the inverse Hessian matrix; this approach leads to a significant reduction on memory use. Nevertheless, it has a quite strong dependence on the initial guess. The Basin-hopping is a stochastic algorithm that usually provides a better approximation of the global maximum. The algorithm iteratively chooses an initial guess at random, proceeds to the local minimization and finally compares the new coordinates with the best ones found so far. This algorithm is however much more time-consuming.

To begin, consider a fairly standard dataset that describes diagnoses of cardiac Single Proton Emission Computed Tomography (SPECT) images [12]. The training dataset contains 80 instances, while the testing dataset contains 187 instances. Examples have 23 binary attributes and there is no missing data. The learning algorithm was tested with the same optimization methods and local structure learning approaches. We compare results obtained for two different local structure learning approaches: (1) accepting only combinations of rules that encode complete probability tables and (2) or any combination of rules. Results obtained are listed in Table 2.

We observe the significant reduction of the number of parameters needed for representation. In addition, results obtained with both optimization algorithms are the exactly same.

We then present results with data generated from a (simulated) faulty Boolean circuit. The purpose of this experiment is to investigate whether our methods

Table 2. Heart Diagnosis experiments.

L-BFGS							
Training set	Testing set	Complete CPT			Any combination of rules		
# Instances	# Instances	MDL	Log-Likelihood	Parameters	MDL	Log-Likelihood	Parameters
80	187	-1341.73	-1281.78	63	-1316.18	-1263.85	55
Basin-hopping							
Training set	Testing set	Complete CPT			Any combination of rules		
# Instances	# Instances	MDL	Log-Likelihood	Parameters	MDL	Log-Likelihood	Parameters
80	187	-1341.73	-1281.78	63	-1316.18	-1263.85	55

Table 3. Binary Adder experiments.

L-BFGS							
Training set	Testing set	Complete CPT			Any combination of rules		
# Instances	# Instances	MDL	Log-Likelihood	Parameters	MDL	Log-Likelihood	Parameters
30	10000	-317635.87	-317590.82	61	-190628.02	-190592.57	48
60	10000	-282916.56	-282860.54	63	-211580.75	-211535.40	51
90	10000	-231156.16	-231096.56	61	-200133.73	-200086.83	48
120	10000	-281634.57	-281571.16	61	-197082.89	-197029.87	51
250	10000	-244964.95517	-244887.02	65	-251550.34	-251489.19	51
500	10000	-228706.11	-228617.04	66	-217679.01	-217608.84	52
1000	10000	-188356.10	-188236.10	80	-177142.65	-177049.65	62
Basin-hopping							
Training set	Testing set	Complete CPT			Any combination of rules		
# Instances	# Instances	MDL	Log-Likelihood	Parameters	MDL	Log-Likelihood	Parameters
30	10000	-344625.96	-344580.91	61	-190687.47	-190652.02	48
1000	10000	-188356.10	-188236.10	80	-177142.52	-177049.52	62

can capture nearly-deterministic systems with less parameters than a typical Bayesian network. We should expect so: rules can encode deterministic relationships compactly, so they should lead to a reduction in the number of necessary parameters.

We simulated a digital circuit for addition in the binary numeral system. We consider the addition of two 4-bit numbers and, therefore, 24 logic gates (XOR, OR, AND). The circuit is used to generate binary datasets, where attributes correspond to the gates outputs. The adder input values are randomly chosen and each gate is associated with a 1% probability of failure, i.e. the likelihood that a gate outputs 0 or 1 at random. Training datasets contain 30, 60, 90, 120, 250, 500 and 1000 instances, while testing datasets contain 10000 instances in all cases. The learning algorithm is tested with both L-BFGS and Basin-hopping optimization methods. Results obtained are listed in Table 3.

We note L-BFGS and Basin-hopping optimization methods perform fairly similar. However, as Basin-hopping is much more time consuming, most tests are run with L-BFGS. For smaller datasets, the algorithm proposed in this paper scores better and requires fewer parameters. For larger datasets, both approaches tend to converge in terms of score, but there is still a significant reduction on the number of parameters.

7 Conclusion

We have described techniques that can learn a PLP from a complete dataset by exact score maximization. Despite the attention that has been paid to PLPs in recent years, it does not seem that exact score-based learning has been attempted so far. This paper offers initial results on such an enterprise; the main contribution is to present cases where closed-form solutions are viable.

The techniques proposed in this paper apply to a restricted albeit powerful class of PLPs. In essence, we have shown that for this class it is possible to maximize the MDL score using a host of insights that simplify computation.

The class we have focused on is the class of acyclic propositional PLPs where rules have at most two literals in their bodies. As acyclic propositional PLPs are closely related to Bayesian networks, we were able to bring results produced for Bayesian network learning into the challenging task of learning probabilistic programs. However, PLPs have features of their own; an advantage is that they can capture conditional distributions with less parameters than usual CPTs; a disadvantage is that learning requires more complex optimization. Our results identify a powerful class of PLPs for which the complexity of optimization can be kept under control.

We have also implemented and tested our methods. We have shown that learned PLPs contain less parameters than the correponding CPT-based Bayesian networks, as intuitively expected. Whenever the model is nearly deterministic, the expressive power of rules leads to improved accuracy.

In future work we intend to extend our techniques to relational but still acyclic programs, and finally to relational and even cyclic programs. For those cases non-trivial extensions will have to be developed as the direct relationship with Bayesian network learning will be lost.

Acknowledgement

The first author is supported by a scholarship from Toshiba Corporation. The second and third authors are partially supported by CNPq.

References

1. Elena Bellodi, and Fabrizio Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming*, 15(2):169–212, 2015.
2. Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Conference on Uncertainty in Artificial Intelligence*, pages 115–123, 1996.
3. Cassio De Campos, and Qiang Ji. Efficient learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, 12:663-689, 2011.
4. James Cussens. Bayesian network learning with cutting planes. In *Conference on Uncertainty in Artificial Intelligence*, pages 153–160, 2011.
5. Evgeny Dantsin, Thomas Eiter, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
6. Luc De Raedt, and Ingo Thon. Probabilistic rule learning. In *Inductive Logic Programming*, volume 6489 of LNCS, pages 47-58, 2010.
7. Luc De Raedt, Anton Dries, Ingo Thon, Guy Van den Broeck, and Mathias Verbeke. Inducing probabilistic relational rules from probabilistic examples. In *International Joint Conference on Artificial Intelligence*, 2015.

8. Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shrerionov, Bernd Gutmann, Gerda Janssens, and Luc de Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2014.
9. Nir Friedman and Moises Goldszmidt. Learning Bayesian networks with local structure. In *Conference on Uncertainty in Artificial Intelligence*, pages 252–262, 1998.
10. Norbert Fuhr. Probabilistic Datalog — a logic for powerful retrieval methods. In *Conference on Research and Development in Information Retrieval*, pages 282–290, Seattle, Washington, 1995.
11. Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.
12. M. Lichman. UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2013.
13. Thomas Lukasiewicz. Probabilistic logic programming. In *European Conference on Artificial Intelligence*, pages 388–392, 1998.
14. Raymond Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.
15. Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.
16. David Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.
17. David Poole. The Independent Choice Logic and beyond. In *Probabilistic Inductive Logic Programming*, volume 4911 of *Lecture Notes in Computer Science*, pages 222–243. Springer, 2008.
18. Luc De Raedt. *Logical and Relational Learning*. Springer, 2008.
19. Fabrizio Riguzzi, Elena Bellodi, and Riccardo Zese. A history of probabilistic inductive logic programming. *Frontiers in Robotics and AI*, 1:1–5, 2014.
20. Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Int. Conference on Logic Programming*, pages 715–729, 1995.
21. Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.
22. Peter Van Beek, and Heilla-Franziska Hoffmann. Machine learning of Bayesian networks using constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 429–445, 2015.
23. David J. Wales, Jonathan P. K. Doye. Global optimization by Basin-Hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms. *Journal of Physical Chemistry A*, 101, 1997.
24. David J. Wales. *Energy Landscapes*. Cambridge University Press, 2003.
25. Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base completion. Technical Report arxiv.org/abs/1702.08367, Carnegie Mellon University, 2017.