Towards Combining Inductive Logic Programming with Bayesian Networks

Kristian Kersting and Luc De Raedt

Institute for Computer Science, Machine Learning Lab
Albert-Ludwigs-University, Georges-Köhler-Allee, Gebäude 079,
D-79085 Freiburg i. Brg., Germany
{kersting,deraedt}@informatik.uni-freiburg.de

Abstract. Recently, new representation languages that integrate first order logic with Bayesian networks have been developed. Bayesian logic programs are one of these languages. In this paper, we present results on combining *Inductive Logic Programming* (ILP) with Bayesian networks to learn both the qualitative and the quantitative components of Bayesian logic programs. More precisely, we show how to combine the ILP setting *learning from interpretations* with score-based techniques for learning Bayesian networks. Thus, the paper positively answers Koller and Pfeffer's question, whether techniques from ILP could help to learn the logical component of first order probabilistic models.

1 Introduction

In recent years, there has been an increasing interest in integrating probability theory with first order logic. One of the research streams [24, 22, 11, 6, 14] aims at integrating two powerful and popular knowledge representation frameworks: Bayesian networks [23] and first order logic. In 1997, Koller and Pfeffer [16] address the question "*where do the numbers come from?*" for such frameworks. At the end of the same paper, they raise the question whether techniques from inductive logic programming (ILP) could help to learn the logical component of first order probabilistic models. In [15] we suggested that the ILP setting *learning from interpretations* [4, 5, 1] is a good candidate for investigating this question. With this paper we would like to make our suggestions more concrete. We present a novel scheme to learn intensional clauses within Bayesian logic programs [13, 14]. It combines techniques from ILP with techniques for learning Bayesian networks. More exactly, we will show that the *learning from interpretations* setting for ILP can be integrated with score-based Bayesian network learning techniques for learning Bayesian logic programs. Thus, we positively answer Koller and Pfeffer's question.

We proceed as follows. After briefly reviewing the framework of Bayesian logic programs in Section 2, we dicuss our learning approach in Section 3. We define the learning problem, introduce the scheme of the algorithm, and discuss it applied on a special class of propositional Bayesian logic programs, well-known under the name Bayesian networks, and applied on general Bayesian logic programs. Before concluding the paper, we relate our approach to other work in Section 5. We assume some familiarity with logic programming or Prolog (see e.g. [26, 18]) as well as with Bayesian networks (see e.g. [23, 2]).

2 Bayesian Logic Programs

Throughout the paper we will use an example from genetics which is inspired by Friedman et al. [6]: "it is a genetic model of the inheritance of a single gene that determines a person's `X` blood type `bt(X)`. Each person `X` has two copies of the chromosome containing this gene, one, `mc(Y)`, inherited from her mother `m(Y,X)`, and one, `pc(Z)`, inherited from her father `f(Z,X)`." We will use $\mathbf{P}$ to denote a probability distribution, e.g. $\mathbf{P}(x)$, and the normal letter P to denote a probability value, e.g. $P(x = v)$, where v is a state of x.

The Bayesian logic program framework we will use in this paper is based on the Datalog subset of definite clausal logic, i.e. no functor symbols are allowed. The idea is that each Bayesian logic program specifies a Bayesian network, with one node for each (Bayesian) ground atom (see below). For a more expressive framework based on pure Prolog we refer to [14].

A Bayesian logic program B consist of two components, firstly a *logical* one, a set of Bayesian clauses (cf. below), and secondly a *quantitative* one, a set of conditional probability distributions and combining rules (cf. below) corresponding to that logical structure. A *Bayesian (definite) clause* c is an expression of the form

$$A \mid A_1, \ldots, A_n$$

where $n \geq 0$, the $A, A_1, \ldots, A_n$ are Bayesian atoms and all Bayesian atoms are (implicitly) universally quantified. We define $\mathit{head}(c) = A$ and $\mathit{body}(c) = \{A_1, \ldots, A_n\}$. So, the differences between a *Bayesian clause* and a *logical* one are : (1) the atoms $p(t_1, ..., t_n)$ and predicates p arising are Bayesian, which means that they have an associated (finite) domain[1] $\mathrm{dom}(p)$, and (2) we use " | " instead of ":-". For instance, consider the Bayesian clause c

```
bt(X) | mc(X), pc(X).
```

where $\mathrm{dom}(\mathit{bt}) = \{a, b, ab, 0\}$ and $\mathrm{dom}(\mathit{mc}) = \mathrm{dom}(\mathit{pc}) = \{a, b, 0\}$. It says that the blood type of a person X depends on the inherited genetical information of X. Note that the domain $\mathrm{dom}(p)$ has nothing to do with the notion of a domain in the logical sense. The domain $\mathrm{dom}(p)$ defines the states of random variables. Intuitively, a Bayesian predicate p generically represents a set of (finite) random variables. More precisely, each Bayesian ground atom g over p represents a (finite) random variable over the states $\mathrm{dom}(g) := \mathrm{dom}(p)$. E.g. $\mathit{bt}(\mathit{ann})$ represents

[1] For the sake of simplicity we consider finite random variables, i.e. random variables having a finite set dom of states. However, the ideas generalize to discrete and continuous random variables.

the blood type of a person named Ann as a random variable over the states $\{a, b, ab, 0\}$. Apart from that, most other *logical* notions carry over to Bayesian logic programs. So, we will speak of Bayesian predicates, terms, constants, substitutions, ground Bayesian clauses, Bayesian Herbrand interpretations etc. We will assume that all Bayesian clauses are range-restricted. A clause is *range-restricted* iff all variables occurring in the head also occur in the body. Range restriction is often imposed in the database literature; it allows one to avoid derivation of non-ground true facts.
In order to represent a probabilistic model we associate with each Bayesian clause c a conditional probability distribution $cpd(c)$ encoding $\mathbf{P}(head(c) \mid body(c))$. To keep the expositions simple, we will assume that $cpd(c)$ is represented as table, see Figure 1. More elaborate representations like decision trees or rules are also possible. The distribution $cpd(c)$ generically represents the conditional probability distributions of all ground instances $c\theta$ of the clause c. In general, one may have many clauses, e.g. clauses c_1 and the c_2

```
bt(X) | mc(X).
bt(X) | pc(X).
```

and corresponding substitutions θ_i that ground the clauses c_i such that $head(c_1\theta_1) = head(c_2\theta_2)$. They specify $cpd(c_1\theta_1)$ and $cpd(c_2\theta_2)$, but not the distribution required: $\mathbf{P}(head(c_1\theta_1) \mid body(c_1) \cup body(c_2))$. The standard solution to obtain the distribution required are so called *combining rules*; functions which map finite sets of conditional probability distributions $\{\mathbf{P}(A \mid A_{i1}, \ldots, A_{in_i}) \mid i = 1, \ldots, m\}$ onto one (*combined*) conditional probability distribution $\mathbf{P}(A \mid B_1, \ldots, B_k)$ with $\{B_1, \ldots, B_k\} \subseteq \bigcup_1^m \{A_{i1}, \ldots, A_{in_i}\}$. We assume that for each Bayesian predicate p there is a corresponding combining rule cr, such as noisy -or.
To summarize, a *Bayesian logic program B* consists of a (finite) set of Bayesian clauses. To each Bayesian clause c there is exactly one conditional probability distribution $cpd(c)$ associated, and for each Bayesian predicate p there is exactly one associated combining rule $cr(p)$.

The declarative semantics of Bayesian logic programs is given by the annotated *dependency graph*. The *dependency graph $DG(B)$* is that directed graph whose nodes correspond to the ground atoms in the least Herbrand model $\mathrm{LH}(B)$ (cf. below). It encodes the *directly influenced by* relation over the random variables in $\mathrm{LH}(B)$: there is an edge from a node x to a node y if and only if there exists a clause $c \in B$ and a substitution θ, s.t. $y = head(c\theta)$, $x \in body(c\theta)$ and for all atoms z appearing in $c\theta : z \in \mathrm{LH}(B)$. The direct predecessors of a graph node x are denoted as its parents, $\mathbf{Pa}(x)$. The Herbrand base $\mathrm{HB}(B)$ is the set of all random variables we could talk about. It is defined as if B were a logic program (cf. [18]). The least Herbrand model $\mathrm{LH}(B) \subseteq \mathrm{HB}(B)$ consists of all *relevant* random variables, the random variables over which a probability distribution is defined by B, as we will see. Again, $\mathrm{LH}(B)$ is defined as if B were be a logic program (cf. [18]). It is the least fix point of the *immediate consequence operator* applied on the empty interpretation. Therefore, a ground atom which is true in the logical sense corresponds to a relevant random variables. Now,

```
m(ann,dorothy).
f(brian,dorothy).
pc(ann).
pc(brian).
mc(ann).
mc(brian).

mc(X)   | m(Y,X),mc(Y),pc(Y).
pc(X)   | f(Y,X),mc(Y),pc(Y).
bt(X)   | mc(X),pc(X).
```
(1)

$mc(X)$	$pc(X)$	$\mathbf{P}(bt(X))$
a	a	$(0.97, 0.01, 0.01, 0.01)$
b	a	$(0.01, 0.01, 0.97, 0.01)$
$\cdots$	$\cdots$	$\cdots$
0	0	$(0.01, 0.01, 0.01, 0.97)$

(2)

Fig. 1. (1) The Bayesian logic program *bloodtype* encoding our genetic domain. To each Bayesian predicate, the identity is associated as combining rule. (2) A conditional probability distribution associated to the Bayesian clause `bt(X) | mc(X), pc(X)` represented as a table.

to each node x in $DG(B)$ the combined conditional probability distribution is associated which is the result of the combining rule $cr(p)$ of the corresponding Bayesian predicate p applied on the set of $cpd(c\theta)$'s where $head(c\theta) = x$ and $\{x\} \cup body(c\theta) \subseteq \mathrm{LH}(B)$. Thus, if $DG(B)$ is acyclic and not empty then it would encode a Bayesian network, because Datalog programs have a finite least Herbrand model which always exists and is unique. Therefore, the following independency assumption holds: *each node x is independent of its non-descendants given a joint state of its parents $\mathbf{Pa}(x)$ in the dependency graph.* E.g. the program in Figure 1 renders $bt(dorothy)$ independent from $pc(brian)$ given a joint state of $pc(dorothy), mc(dorothy)$. Using this assumption the following proposition is provable:

Proposition 1. *Let B be a Bayesian logic program. If B fulfills that*

1. $\mathrm{LH}(B) \neq \emptyset$ *and*
2. $DG(B)$ *is acyclic*

then it specifies a unique probability distribution $\mathbf{P}_B$ over $\mathrm{LH}(B)$.

To see this, remember that if the conditions are fulfilled then $DG(B)$ is a Bayesian network. Thus, given a total order $x_1 \ldots, x_n$ of the nodes in $DG(B)$ the distribution P_B factorizes in the usual way: $\mathbf{P}_B(x_1 \ldots, x_n) = \prod_{i=1}^{n} \mathbf{P}(x_i \mid \mathbf{Pa}\, x_i)$, where $\mathbf{P}(x_i \mid \mathbf{Pa}\, x_i)$ is the combined conditional probability distribution associated to x_i. A program B fulfilling the conditions is called *well-defined*, and we will consider such programs for the rest of the paper. The program *bloodtype* in Figure 1 encodes the regularities in our genetic example. Its grounded version, which is a Bayesian network, is given in Figure 2. This illustrates that Bayesian networks [23] are well-defined propositional Bayesian logic programs. Each node-parents pair uniquely specifies a propositional Bayesian clause; we associate the identity as combining rule to each predicate; the conditional probability distributions are the ones of the Bayesian network.

```
m(ann,dorothy).
f(brian,dorothy).
pc(ann).
pc(brian).
mc(ann).
mc(brian).
mc(dorothy)  | m(ann, dorothy),mc(ann),pc(ann).
pc(dorothy)  | f(brian, dorothy),mc(brian),pc(brian).
bt(ann)      | mc(ann), pc(ann).
bt(brian)    | mc(brian), pc(brian).
bt(dorothy)  | mc(dorothy),pc(dorothy).
```

Fig. 2. The grounded version of the Bayesian logic program of Figure 1. It (directly) encodes a Bayesian network.

3 Structural Learning of Bayesian Logic Programs

Let us now focus on the logical structure of Bayesian logic programs. When designing Bayesian logic programs, the expert has to determine this (logical) structure of the Bayesian logic program by specifying the extensional and intensional predicates, and by providing definitions for each of the intensional predicates. Given this logical structure, the Bayesian logic program induces (the structure of) a Bayesian network whose nodes are the *relevant*[2] random variables. It is well-known that determining the structure of a Bayesian network, and therefore also of a Bayesian logic program, can be difficult and expensive. On the other hand, it is often easier to obtain a set $D = \{D_1, \ldots, D_m\}$ of data cases. A data case $D_i \in D$ has two parts, a logical and a probabilistic part.

The logical part of a data case $D_i \in D$, denoted as $Var(D_i)$, is a Herbrand interpretation. Consider e.g. the least Herbrand model $LH(bloodtype)$ (cf. Figure 2) and the logical atoms $LH(bloodtype')$ in the following case:

$$\{m(cecily, fred), f(henry, fred), pc(cecily), pc(henry), pc(fred),$$
$$mc(cecily), mc(henry), mc(fred), bt(cecily), bt(henry), bt(fred)\}$$

These (logical) interpretations can be seen as the least Herbrand models of unknown Bayesian logic programs. They specify different sets of *relevant* random variables, depending on the given "extensional context". If we accept that the genetic laws are the same for both families then a learning algorithm should transform such extensionally defined predicates into intensionally defined ones, thus compressing the interpretations. This is precisely what ILP techniques are doing. The key assumption underlying any inductive technique is that the rules that are valid in one interpretation are likely to hold for any interpretation.

[2] In a sense, *relevant* random variables are those variables, which Cowell et al. [2, p. 25] mean when they say that the first phase in developing a Bayesian network involves to "*specify the set of 'relevant' random variables*".

It thus seems clear that techniques for *learning from interpretations* can be adapted for learning the logical structure of Bayesian logic programs. *Learning from interpretations* is an instance of the non-monotonic learning setting of ILP (cf. [19]), which uses only only positive examples (i.e. models).

So far, we have specified the logical part of the learning problem: we are looking for a set H of Bayesian clauses given a set D of data cases s.t. $\forall D_i \in D : \mathrm{LH}(H \cup \mathit{Var}(D_i)) = \mathit{Var}(D_i)$, i.e. the Herbrand interpretation $\mathit{Var}(D_i)$ is a model for H. The hypotheses H in the space $\mathcal{H}$ of hypotheses are sets of Bayesian clauses. However, we have to be more careful. A candidate set $H \in \mathcal{H}$ has to be acyclic on the data that means that for each $D_i \in D$ the induced Bayesian network over $\mathrm{LH}(H \cup \mathit{Var}(D_i))$ has to be acyclic. Let us now focus on the quantitative components. The quantitative component of a Bayesian logic program is given by the associated conditional probability distributions and combining rules. We assume that the combining rules are fixed. Each data case $D_i \in D$ has a probabilistic part which is a partial assignment of states to the random variables in $\mathit{Var}(D_i)$. We say that D_i is a *partially observed joint state* of $\mathit{Var}(D_i)$. As an example consider the following two data cases:

$$\{m(\mathit{cecily}, \mathit{fred}) = \mathit{true}, f(\mathit{henry}, \mathit{fred}) = ?, \mathit{pc}(\mathit{cecily}) = a, \mathit{pc}(\mathit{henry}) = b, \mathit{pc}(\mathit{fred}) = ?,$$
$$\mathit{mc}(\mathit{cecily}) = b, \mathit{mc}(\mathit{henry}) = b, \mathit{mc}(\mathit{fred}) = ?, \mathit{bt}(\mathit{cecily}) = \mathit{ab}, \mathit{bt}(\mathit{henry}) = b, \mathit{bt}(\mathit{fred}) = ?\}$$

$$\{m(\mathit{ann}, \mathit{dorothy}) = \mathit{true}, f(\mathit{brian}, \mathit{dorothy}) = \mathit{true}, \mathit{pc}(\mathit{ann}) = b,$$
$$\mathit{mc}(\mathit{ann}) = ?, \mathit{mc}(\mathit{brian}) = a, \mathit{mc}(\mathit{dorothy}) = a, \mathit{pc}(\mathit{dorothy}) = a,$$
$$\mathit{pc}(\mathit{brian}) = ?, \mathit{bt}(\mathit{ann}) = \mathit{ab}, \mathit{bt}(\mathit{brian}) = ?, \mathit{bt}(\mathit{dorothy}) = a\},$$

where ? denotes an unknown state of a random variable. The partial assignments induce a joint distribution over the random variables of the logical parts. A candidate $H \in \mathcal{H}$ should reflect this distribution. In Bayesian networks the conditional probability distributions are typically learned using gradient descent or EM for a fixed structure of the Bayesian network. A scoring mechanism that evaluates how well a given structure $H \in \mathcal{H}$ matches the data is maximized. Therefore, we will assume a function $\mathit{score}_D : \mathcal{H} \mapsto \mathbb{R}$.

To summarize, the learning problem can be formulated as follows:

Given a set $D = \{D_1, \ldots, D_m\}$ of data cases, a set $\mathcal{H}$ of Bayesian logic programs and a scoring function $\mathit{score}_D : \mathcal{H} \mapsto \mathbb{R}$.

Find a candidate $H^* \in \mathcal{H}$ which is acyclic on the data such that for all $D_i \in D$: $\mathrm{LH}(H^* \cup \mathit{Var}(D_i)) = \mathit{Var}(D_i)$, and H^* matches the data D best according to score_D.

The best match in this context refers to those parameters of the associated conditional probability distributions which maximize the scoring function. For a discussion on how the best match can be computed see [12] or [16]. The chosen scoring function is a crucial aspect of the algorithm. Normally, we can only hope to find a sub-optimal candidate. A heuristic learning algorithm solving this problem is given in Algorithm 1.

$\mathbf{P}(A \mid A_1, \ldots, A_n)$					
true	*false*	A_1	A_2	$\ldots$	A_n
1.0	0.0	*true*	*true*		*true*
0.0	1.0	*false*	*true*		*true*
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
0.0	1.0	*false*	*false*		*false*

Table 1. The conditional probability distribution associated to a Bayesian clause $\mathtt{A} \mid \mathtt{A}_1, \ldots, \mathtt{A}_n$ encoding a logical one.

Background knowledge can be incorporated in our approach in the following way. The background knowledge can be expressed as a fixed Bayesian logic program B. Then we search for a candidate H^* which is together with B acyclic on the data such that for all $D_i \in D : \mathrm{LH}(B \cup H^* \cup \mathit{Var}(D_i)) = \mathit{Var}(D_i)$, and $B \cup H^*$ matches the data D best according to score_D. In [14], we show how pure Prolog programs can be repesented as Bayesian logic prorgams w.r.t. the conditions 1 and 2 of Proposition 1. The basic idea is as follows. Assume that a logical clause $\mathtt{A} : -\mathtt{A}_1, \ldots, \mathtt{A}_n$ is given. We encode the clause by the Bayesian clause $\mathtt{A} : -\mathtt{A}_1, \ldots, \mathtt{A}_n$ where $A, A_1, \ldots, A_n$ are now Bayesian atoms over $\{\mathit{true}, \mathit{false}\}$. We associate to the Bayesian clause the conditional probability distribution of Figure 1, and set the combining rule of A's predicate to *max*:

$$\begin{aligned}&\max\{\mathbf{P}(A \mid A_{i1}, \ldots, A_{in_i}) \mid i = 1, \ldots, n\} = \\ &\qquad \mathbf{P}(A \mid \cup_{i=1}^{n}\{A_{i1}, \ldots, A_{in_i}\}) := \max_{i=1}^{n}\{\mathbf{P}(A \mid A_{i1}, \ldots, A_{in_i})\}.\end{aligned} \tag{1}$$

We will now explain Algorithm 1 and its underlying ideas in more details. The next section illustrates the algorithm for a special class of Bayesian logic programs: Bayesian networks. For Bayesian networks, the algorithm coincides with score-based methods for learning within Bayesian networks which are proven to be useful by the UAI community (see e.g. [9]). Therefore, an extension to the first order case seems reasonable. It will turn out that the algorithm works for first order Bayesian logic programs, too.

3.1 A Propositional Case: Bayesian Networks

Here we will show that Algorithm 1 is a generalization of score-based techniques for structural learning within Bayesian networks. To do so we briefly review these score-based techniques. Let $x = \{x_1, \ldots, x_n\}$ be a fixed set of random variables. The set x corresponds to a least Herbrand model of an unknown propositional Bayesian logic program representing a Bayesian network. The probabilistic dependencies among the relevant random variables are not known, i.e. the propositional Bayesian clauses are unknown. Therefore, we have to select such a propositional Bayesian logic program as a *candidate* and estimate its parameters. The data cases of the data $D = \{D_1, \ldots, D_m\}$ look like

```
Let $H$ be an initial (valid) hypothesis;
$S(H) := score_D(H)$;
repeat
    $H' := H$;
    $S(H') := S(H)$;
    foreach $H'' \in \rho_g(H') \cup \rho_s(H')$ do
        if $H''$ is (logically) valid on $D$ then
            if the Bayesian networks induced by $H''$ on the data are acyclic
            then
                if $score_D(H'') > S(H)$ then
                    $H := H''$;
                    $S(H) := S(H'')$;
                end
            end
        end
    end
until $S(H') = S(H)$;
Return $H$;
```

Algorithm 1: A greedy algorithm for searching the structure of Bayesian logic programs.

$$\{m(ann, dorothy) = true, f(brian, dorothy) = true, pc(ann) = a,$$
$$mc(ann) =?, mc(brian) =?, mc(dorothy) = a, mc(dorothy) = a,$$
$$pc(brian) = b, bt(ann) = a, bt(brian) =?, bt(dorothy) = a\}$$

which is a data case for the Bayesian network in Figure 2. Note, that the atoms have to be interpreted as propositions. The set of candidate Bayesian logic programs spans the hypothesis space $\mathcal{H}$. Each $H \in \mathcal{H}$ is a Bayesian logic program consisting of n propositional clauses: for each $x_i \in x$ a single clause c with $head(c) = x_i$ and $body(c) \subseteq x \setminus \{x_i\}$. To traverse $\mathcal{H}$ we (1) specify two *refinement* operators $\rho_g : \mathcal{H} \mapsto 2^{\mathcal{H}}$ and $\rho_s : \mathcal{H} \mapsto 2^{\mathcal{H}}$, that take a candidate and modify it to produce a set of candidates. The search algorithm performs informed search in $\mathcal{H}$ based on $score_D$. In the case of Bayesian networks the operator $\rho_g(H)$ deletes a Bayesian proposition from the body of a Bayesian clause $c_i \in H$, and the operator $\rho_s(H)$ adds a Bayesian proposition to the body of $c_i \in H$. Usually, instances of scores are e.g. the *minimum description length* score [17] or the Bayesian scores [10].

As a simple illustration we consider a greedy hill-climbing algorithm incorporating $score_D(H) := LL(D, H)$, the log-likelihood of the data D given a candidate structure H with the best parameters. We pick an initial candidate $S \in \mathcal{H}$ as starting point (e.g. the set of all propositions) and compute the likelihood $LL(D, S)$ with the best parameters. Then, we use $\rho(S)$ to compute the legal "neighbours" (candidates being acyclic) of S in $\mathcal{H}$ and score them. All neighbours

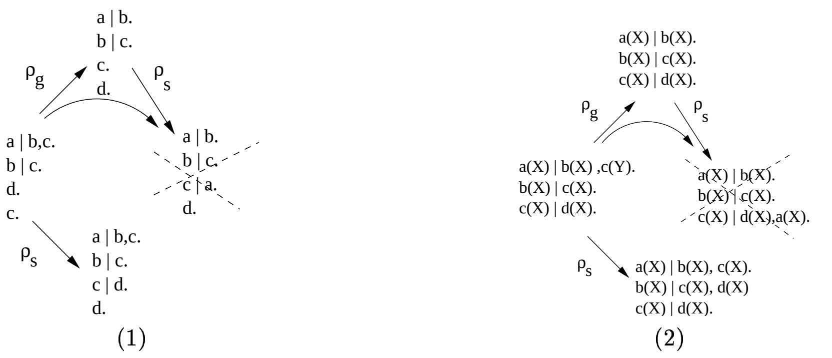

Fig. 3. (1) The use of refinement operators during structural search for Bayesian networks. We can add (ρ_s) a proposition to the body of a clause or delete (ρ_g) it from the body. (2) The use of refinement operators during structural search within the framework of Bayesian logic programs. We can add (ρ_s) a constant-free atom to the body of a clause or delete (ρ_g) it from the body. Candidates crossed out in (1) and (2) are illegal because they are cyclic.

are valid (see below for a definition of validity). E.g. replacing `pc(dorothy)` with `pc(dorothy) | pc(brian)` gives such a "neighbour". We take that $S' \in \rho(S)$ with the best improvements in the score. The process is continued until no improvements in score are obtained. The use of the two refinement operators is illustrated in Figure 3.

3.2 The First Order Case

Here, we will explain the ideas underlying our algorithm in the first order case. On the logical level it is similar to the ILP setting *learning from interpretation* which e.g. is used in the CLAUDIEN system ([4, 5, 1]): (1) all data cases are interpretations, and (2) a hypothesis should reflect what is in the data. The first point is carried over by enforcing each data case $D_i \in \{D_1, \ldots, D_m\}$ to be a partially observed joint state of a Herbrand interpretation of an unknown Bayesian logic program. This also implies that all data cases are probabilistically independent[3]. The second point is enforced by requiring all hypotheses to be (logically) true in all data cases, i.e. the logical structure of the hypothesis is certain. Thus, the logical rules valid on the data cases are constraints on the space of hypotheses. The main difference to the pure logical setting is that we have to take the probabilistic parts of the data case into account.

Definition 1 (Characteristic induction from interpretations). *(adapted w.r.t. our purposes from [5]) Let D be a set of data cases and $\mathcal{C}$ the set of all clauses that can be part of a hypothesis. $H \subseteq \mathcal{C}$ is a logical solution iff H is a logically maximally general valid hypothesis. A hypothesis $H \subseteq \mathcal{C}$ is (logically)*

[3] An assumption which one has to verify if using our method. In the case of families the assumption seems reasonable.

valid iff for all $D_i \in D$: H is (logically) true in D_i. A hypothesis $H \subseteq \mathcal{C}$ is a probabilistic solution iff H is a valid hypothesis and the Bayesian network induced by H on D is acyclic.

It is common to impose syntactic restrictions on the space $\mathcal{H} = 2^{\mathcal{C}}$ of hypotheses through the language $\mathcal{L}$, which determines the set $\mathcal{C}$ of clauses that can be part of a hypothesis. The language $\mathcal{L}$ is an important parameter of the induction task.

Language Assumption. *In this paper, we assume that the alphabet of $\mathcal{L}$ only contains constant and predicate symbols that occur in one of the data cases, and we restrict $\mathcal{C}$ to range-restricted, constant-free clauses containing maximum $k = 3$ atoms in the body. Furthermore, we assume that the combining rules associated to the Bayesian predicates are given.*

Let us discuss some properties of our setting. (1) Using partially observed joint states of interpretations as data cases is the first order equivalent of what is done in Bayesian network learning. There each data case is described by means of a partially observed joint state of a fixed, finite set of random variables. Furthermore, it implicitly corresponds to assuming that all relevant ground atoms of each data case are known: all random variables not stated in the data case are regarded to be *not relevant* (false in the logical sense). (2) Hypotheses have to be valid. Intuitively, validity means that the hypothesis holds (logically) on the data, i.e. that the induced hypothesis postulates true regularities present in the data cases. Validity is a monotone property at the level of clauses, i.e. if H_1 and H_2 are valid with respect to a set of data cases D, then $H_1 \cup H_2$ is valid. This means that all well-formed clauses in $\mathcal{L}$ can (logically) be considered completely independent of each other. Both arguments (1) and (2) together guarantee that no possible dependence among the random variables is lost. (3) The condition of maximal generality appears in the definition because the most interesting hypotheses in the logical case are the most informative and hence the most general. Therefore, we will use a logical solution as initial hypotheses. But the best scored hypothesis has not to be maximally general, as the initial hypothesis in the next example shows. Here, our approach differs from the pure logical setting. We consider probabilistic solutions instead of logical solutions. The idea is to incorporate a scoring function known from learning of Bayesian networks to evaluate how well the given probabilistic solution matches the data.

The key to our proposed algorithm is the well-known definition of logical entailment (cf. [18]). It induces a partial order on the set of hypotheses. To compute our initial (valid) hypotheses we use the CLAUDIEN algorithm. Roughly speaking, CLAUDIEN works as follows (for a detailed discussion we refer to [5]). It keeps track of a list of candidate clauses Q, which is initialized to the maximally general clause (in $\mathcal{L}$). It repeatedly deletes a clause c from Q, and tests whether c is valid on the data. If it is, c is added to the final hypothesis, otherwise, all maximally general specializations of c (in $\mathcal{L}$) are computed (using a so-called refinement operator ρ, see below) and added back to Q. This process continues until Q is empty and all relevant parts of the search-space have been considered. We now have to define operators to traverse $\mathcal{H}$. A logical specialization (or generalization) of a set H of Bayesian clauses could be achieved by specializing (or generalizing) single clauses $c \in H$. In our approach we use the two refinement operators $\rho_s : 2^{\mathcal{H}} \mapsto \mathcal{H}$ and $\rho_g : 2^{\mathcal{H}} \mapsto \mathcal{H}$. The operator $\rho_s(H)$ adds constant-free atoms to the body of a single clause $c \in H$, and $\rho_g(H)$ deletes constant-free atoms from the body of a single clause $c \in H$. Figure 3 shows the different refinement operators for the general first order case and the propositional case for learning Bayesian networks. Instead of adding (deleting) propositions to (from) the body of a clause, they add (delete) according to our language assumption constant-free atoms. Furthermore, Figure 3 shows that using the refinement operators each probabilistic solution could be reached.

As a simple instantiation of Algorithm 1 we consider a greedy hill-climbing algorithm incorporating $score_D(H) := LL(D, H)$. It picks up a (logical) solution $S \in \mathcal{H}$ as starting point and computes $LL(D, S)$ with the best parameters. For a discussion of how these parameters can be found we refer to [12, 16]. E.g. having data cases over $LH(bloodtype)$ and $LH(bloodtype')$, we choose as initial candidate

```
mc(X)   | m(Y, X).
pc(X)   | f(Y, X).
bt(X)   | mc(X).
```

It is likely that the initial candidate is not a probabilistic solution, although it is a logical solution. E.g. the blood type does not depend on the fatherly genetical information. Then, we use $\rho_s(S)$ and $\rho_g(S)$ to compute the legal "neighbours" of S in $\mathcal{H}$ and score them. E.g. one such a "neighbour" is given by replacing `bt(X) | mc(X)` with `bt(X) | mc(X), pc(X)`. Let S' be that valid and acyclic neighbour which is scored best. If $LL(D, S) < LL(D, S')$, then we take S' as new hypothesis. The process is continued until no improvements in score are obtained. During the search we have to take care to prune away every hypothesis H which is invalid or leads to cyclic dependency graphs (on the data cases). This could be tested in time $O(s \cdot r^3)$ where r is the number of random variables of the largest data case in D and s is the number of clauses in H. To do so, we build the Bayesian networks induced by H over each $Var(D_i)$ by computing the ground instances for each clause $c \in H$ where the ground atoms are members of $Var(D_i)$. This takes $O(s \cdot r_i^3)$. Then, we test in $O(r_i)$ for a topological order of the nodes in the induced Bayesian network.

4 Preliminary Experiments

We have implemented the algorithm in Sicstus Prolog 3.8.1. The implementation has an interface to Matlab to score hypotheses using the BNT toolbox [21]. We considered two totally independent families using the predicates given by *bloodtype* having 12 respectively 15 family members. For each least Herbrand model 1000 samples from the induced Bayesian network were gathered.

The general question was whether we could learn the intensional rules of *bloodtype*. Therefore, we first had a look at the (logical) hypotheses space. The space could be seen as the first order equivalent of the space for learning the structure of Bayesian networks (see Figure 3). In a further experiment the goal was to learn a definition for the predicate *bt*. We had fixed the definitions for the other predicates in two ways: (1) to the definitions the CLAUDIEN system had computed, and (2) to the definitions from the *bloodtype* Bayesian logic program. In both cases, the algorithm scored `bt(X) | mc(X), pc(X)` best, i.e. the algorithm has re-discovered the intensional definition which was originally used to build the data cases. Furthermore, the result shows that the best scored solution was independent of the fixed definitions. This could indicate that ideas about decomposable scoring functions can or should be lifted to the first order case. Although, these experiments are preliminary, they suggest that ILP techniques can be adapted for structural learning within first order probabilistic frameworks.

5 Related Work

To the best of our knowledge, there has not been much work on learning within first order extensions of Bayesian networks. Koller and Pfeffer [16] show how to estimate the maximum likelihood parameters for Ngo and Haddawys's framework of probabilistic logic programs [22] by adapting the EM algorithm. Kersting and De Raedt [12] discuss a gradient-based method to solve the same problem for Bayesian logic programs. Friedman et al. [6, 7] tackle the problem of learning the logical structure of first order probabilistic models. They used Structural-EM for learning probabilistic relational models. This algorithm is similar to the standard EM method except that during iterations of this algorithm the structure is improved. As far as we know this approach, it does not consider logical constraints on the space of hypotheses in the way our approach does. Therefore, we suggest that both ideas can be combined. There exist also methods for learning within first order probabilistic frameworks which do not build on Bayesian networks. Sato et al. [25] give a method for EM learning of PRISM programs. They do not incorporate ILP techniques. Cussens [3] investigates EM like methods for estimating the parameters of stochastic logic programs. Within the same framework, Muggleton [20] uses ILP techniques to learn the logical structure. The used ILP setting is different to *learning from interpretations* and seems not to be based on learning of Bayesian networks.

Finally, Bayesian logic programs are somewhat related to the BUGS language [8]. The BUGS language is based on imperative programming. It uses concepts such as for-loops to model regularities in probabilistics models. So, the differences between Bayesian logic programs and BUGS are akin to the diferences between declarative programming languages (such as Prolog) and imperative ones. Therefore, adapting techniques from Inductive Logic Programming to learn the structure of BUGS programs seems not to be that easy.

6 Conclusions

A new link between ILP and learning within Bayesian networks is presented. We have proposed a scheme for learning the structure of Bayesian logic programs.

It builds on the ILP setting *learning from interpretations*. We have argued that by adapting this setting score-based methods for structural learning of Bayesian networks could be updated to the first order case. The ILP setting is used to define and traverse the space of (logical) hypotheses. Instead of score-based greedy algorithm other UAI methods such as Structural-EM may be used. The experiments we have are promising. They show that our approach works. But the link established between ILP and Bayesian networks seems to be bi-directional. Can ideas developed in the UAI community be carried over to ILP?

The research within the UAI community has shown that score-based methods are useful. In order to see whether this still holds for the first-order case we will perform more detailed experiments. Experiments on real-world scale problems will be conducted. We will look for more elaborated scoring functions like e.g. scores based on the *minimum description length* principle. We will investigate more difficult tasks like learning multiple clauses definitions. The use of refinement operators adding or deleting non constant-free atoms should be explored. Furthermore, it would be interesting to weaken the assumption that a data case corresponds to a complete interpretation. Not assuming all relevant random variables are known would be interesting for learning intensional rules like `nat(s(X)) | nat(X)`. Lifting the idea of decomposable scoring function to the first order case should result in a speeding up of the algorithm. In this sense, we believe that the proposed approach is a good point of departure for further research.

Acknowledgments The authors would like to thank Stefan Kramer and Manfred Jaeger for helpful discussions on the proposed approach. Also, many thanks to the anonymous reviewers for their helpful comments on the initial draft of this paper.

References

1. H. Blockeel and L. De Raedt. ISIDD: An Interactive System for Inductive Databse Design. *Applied Artificial Intelligence*, 12(5):385, 421 1998.
2. R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic networks and expert systems.* Springer-Verlag New York, Inc., 1999.
3. J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 2001. to appear.
4. L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-1993)*, pages 1058–1063, 1993.
5. L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, (26):99–146, 1997.
6. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-1999)*, Stockholm, Sweden, 1999.
7. L. Getoor, D. Koller, B. Taskar, and N. Friedman. Learning probabilistic relational models with structural uncertainty. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, 2000.

8. W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. A language and program for complex bayesian modelling. *The Statistician*, 43, 1994.
9. D. Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Advanced Technology Division, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, March 1995.
10. D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research, 1994.
11. M. Jaeger. Relational Bayesian networks. In *Proceedings of UAI-1997*, 1997.
12. K. Kersting and L. De Raedt. Adaptive Bayesian Logic Programs. In this volume.
13. K. Kersting and L. De Raedt. Bayesian logic programs. In *Work-in-Progress Reports of the Tenth International Conference on Inductive Logic Programming (ILP -2000)*, 2000. `http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-35/`.
14. K. Kersting and L. De Raedt. Bayesian logic programs. Technical Report 151, University of Freiburg, Institute for Computer Science, April 2001.
15. K. Kersting, L. De Raedt, and S. Kramer. Interpreting Bayesian Logic Programs. In *Working Notes of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, 2000.
16. D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-1997)*, pages 1316–1321, Nagoya, Japan, August 23-29 1997.
17. W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10(4), 1994.
18. J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 2. edition, 1989.
19. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.
20. S. H. Muggleton. Learning stochastic logic programs. In L. Getoor and D. Jensen, editors, *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, 2000.
21. K. P. Murphy. *Bayes Net Toolbox for Matlab*. U. C. Berkeley. `http://www.cs.berkeley.edu/~murphyk/Bayes/bnt.html`.
22. L. Ngo and P. Haddawy. Answering queries form context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1997.
23. J. Pearl. *Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2. edition, 1991.
24. D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.
25. T. Sato and Y. Kameya. A viterbi-like algorithm and EM learning for statistical abduction. In *Proceedings of UAI2000 Workshop on Fusion of Domain Knowledge with Data for Decision Support*, 2000.
26. L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. The MIT Press, 1986.