# Adaptive Bayesian Logic Programs

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.** First order probabilistic logics combine a first order logic with a probabilistic knowledge representation. In this context, we introduce *continuous* Bayesian logic programs, which extend the recently introduced Bayesian logic programs to deal with continuous random variables. Bayesian logic programs tightly integrate definite logic programs with Bayesian networks. The resulting framework nicely seperates the qualitative (i.e. logical) component from the quantitative (i.e. the probabilistic) one. We also show how the quantitative component can be learned using a gradient-based maximum likelihood method.

## 1 Introduction

In recent years, there has been an increasing interest in integrating probability theory with first order logic leading to different types of "first order probabilistic logics". One of the streams [22, 20, 11, 15, 12] concentrates on first order extensions of Bayesian networks [21], i.e. it aims at integrating two powerful and popular knowledge representation frameworks: Bayesian networks and first order logic. When investigating the state-of-the-art in this stream (cf. [20, 11, 15, 12]), then there are two important shortcomings of the mentioned techniques. They either do not allow to model continuous[1] random variables or do not use (logical) languages that allow for functor symbols. Nevertheless, both of these features are highly desirable for true "first order probabilistic logics". Indeed, almost every real-world domain, including biology, medicine and finance involves continuous variables, and also, domains involving a potentially infinite number of random variables occur quite naturally in practice (e.g. temporal processes), which requires the use of functors to model the domain.

The first contribution of this paper is the introduction of *continuous* Bayesian logic programs, which allow to model infinite domains using functors as well as continuous random variables. The semantics of these Bayesian logic programs is given in the context of discrete-time stochastic processes. Because, as we have argued in [12], (*discrete*) Bayesian logic programs[2] can serve as a kind of common kernel of first order extensions of Bayesian networks such as probabilistic

[1] We understand in this paper a continuous variable as a variable having $\mathbb{R}$ or a compact interval in $\mathbb{R}$ as domain. A discrete random variable has a countable domain.

[2] Discrete Bayesian logic programs are Bayesian logic programs allowing only for discrete random variables, see [12].

logic-programs[20], relational Bayesian networks [11] and probabilistic relational models [15], *continuous* Bayesian logic programs are novel. They generalize dynamic Bayesian networks, Kalman filters, hidden Markov models, etc.

The second contribution of this paper addresses the famous question: "*where do the numbers, the parameters of the quantitative aspects come from?*". So far, this issue has not yet attracted much attention in the context of first order extensions of Bayesian networks (with the exception of [16, 7] ). In this context, we present for the first time how to calculate the *gradient* for a maximum likelihood estimation of the parameters of Bayesian logic programs. This gives one a rich class of optimization techniques such as conjugate gradient and the possibility to speed up techniques based on the EM algorithm, see [18].

We proceed as follows. After motivating continuous Bayesian logic programs with a simplified example from quantitative genetics in Section 3 we introduce continuous Bayesian logic programs in Section 4. In Section 5 we formulate the likelihood of the parameters of a Bayesian logic program given some data and, based on this, we present in Section 5 a gradient-based method to find that parameters which maximize the likelihood. After discussing related work in Section 7 and reporting experimental experiences in Section 8 we conclude. We assume some familiarity with logic programming (see e.g. [17]) as well as with Bayesian networks [21].

## 2 Quantitative Genetics

A natural domain where Bayesian logic programs should help is *genetics*. Here, the family relationship forms the basis for the dependencies between the random variables and the biological laws provide the probability distribution. Even if the genotype may best be modeled using discrete random variables, some phenotypes such as the height of a person are naturally represented using continuous variables. Moreover, in many situations phenotypes can be influenced by environmental (continuous) quantities such as the amount of nuclear radiation. The subfield of genetics which deals with continuous phenotype is called *quantitative genetics* (cf. [6]).

As an example consider a simplified model of the inheritance of the heights of persons. The height $h(X)$ of a specific person $X$ interpreted as a continuous random variable (having the $\mathrm{dom}(h) = \mathbb{R}$) depends on its genotype $g(X)$, a discrete random variable. The genotype $g(X)$ itself depends on the genotype of the mother $g(M)$ and father $g(F)$. Furthermore, we could assume that $h(X)$ is influenced by the heights of its mother $h(M)$ and father $h(F)$. Figure 1 shows a graph modelling the described dependencies. The graph can be seen as the dependency structure of a Bayesian network [21]. Thus, we are interested in representing the *joint probability density function* (jpdf) $p(g(X), g(M), g(F), h(X), h(M), h(F))$. Let $p$ denote a probability density and $P$ a probability distribution. The chain rule of probability states $p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_{i-1}, \ldots, x_1)$ for a set $\{x_1, \ldots, x_n\}$ of random variables. The known biological dependencies express conditional independency statements such as $g(X)$ is conditional independent

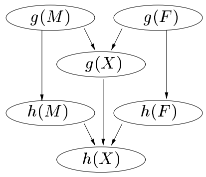


**Fig. 1.** The dependencies of the genetic domain. The height $h(X)$ of a person $X$ depends on the heights $h(M), h(F)$ of its mother $M$ and father $F$.

from $h(X)$ given a joint state of $g(M), g(F)$, i.e. $p(g(X), h(X) \mid g(M), g(F)) = p(g(X) \mid g(M), g(F))$. From the chain rule and the biological laws it follows that

$$\begin{aligned} p(g(X), & g(M), g(F), h(X), h(M), h(F)) = \\ & p(g(X) \mid g(M), g(F)) \cdot p(g(M)) \cdot p(g(F)) \cdot \\ & p(h(X) \mid h(M), h(F)) \cdot p(h(M)) \cdot p(h(F)). \end{aligned} \tag{1}$$

Thus, we must define an infinite set of densities for $h(X)$, one for every possible joint value $\mathbf{u}$ of its parents $\mathbf{Pa}(h(X)) = \{h(M), h(F)\}$ (the direct predecessors of a variable in the dependency graph) can take. Hence, for each $u \in \mathrm{dom}(h(X))$ we have a function $cpd(h(X) \mid \mathbf{Pa}(h(X)))(u \mid \mathbf{u})$ that denotes the conditional probability density $p(h(X) = u \mid \mathbf{Pa}(h(X)) = \mathbf{u})$. We will call such a function a probability density function (pdf). Note, that the used upper types for $X$, $F$ and $M$ do not indicate that they are variables in a logical sense. The representation so far is inherent propositional, i.e. the regularities cannot intensionally be represented. We have to describe it for each "family" $X, M, F$. The framework of *continuous* Bayesian logic programs aims at intensionally representing such regularities.

## 3 (Continuous) Bayesian Logic Programs

A Bayesian logic program $B$ consist of two components, firstly a *logical* one, a set of Bayesian clauses (cf. below) that encodes the assertions of conditional independence in Equation (1), and secondly a *quantitative* one, a set of conditional probability functions and combining rules (cf. below) corresponding to that logical structure. In particular, a *Bayesian (definite) clause* $c$ is an expression of the form

$$A \mid A_1, \ldots, A_n \tag{2}$$

where $n \geq 0$, the $A, A_1, \ldots, A_n$ are Bayesian atoms and all Bayesian atoms are (implicitly) universally quantified. We define $head(c) = A$ and $body(c) = \{A_1, \ldots, A_n\}$. The differences between a *Bayesian* and a *logical* clause are : (1) the atoms $p(t_1, ..., t_m)$ and predicates arising are Bayesian, i.e. they have an associated domain $\mathrm{dom}(p)$, and (2) we use " | " instead of ":-". Furthermore, most other *logical* notions carry over to Bayesian logic programs. So,

```
m(uta,john).
f(peter,john).
g(uta).
g(peter).
g(john)   | m(uta,john),g(uta),f(peter,john),g(peter).
h(uta)    | g(uta).
h(peter) | g(peter).
h(john)   | g(john), m(uta,john),h(uta),f(peter,john),h(peter).
```

**Fig. 2.** A Bayesian logic program which essentially encodes the Bayesian network in Figure 1. Here, Uta and Peter are the parents of John. The Bayesian logic program is the grounded version of the Bayesian logic program in Figure 3.

we will speak of Bayesian predicates, terms, constants, functors, substitutions, ground Bayesian clauses, etc. For instance, consider the Bayesian clause $c_1$ `h(X) | m(X,M), h(M)` where $\mathrm{dom}(m) = \{\mathit{true}, \mathit{false}\}$ and $\mathrm{dom}(h) = \mathbb{R}$. It says that the height of a person $X$ depends on the height of its mother $M$. Intuitively, a Bayesian predicate generically represents a set of random variables. More precisely, each Bayesian ground atom $p(t_1, \ldots, t_m)$ corresponds to a random variable with $\mathrm{dom}(p(t_1, \ldots, t_m)) := \mathrm{dom}(p)$. As long as no ambiguity occurs, we do not distinguish between a Bayesian predicate (atom) and its corresponding logical predicate (atom).

In order to represent a probabilistic model we associate to each Bayesian clause $c$ a probability density function $\mathit{cpd}(c)$ encoding $p(\mathit{head}(c) \mid \mathit{body}(c))$. It generically represents the conditional probability densities of all ground instances $c\theta$ of the clause $c$. In general, one may have many clauses, e.g. $c_1$ and the clause $c_2$ `h(X) | f(X,F),h(F)` and corresponding substitutions $\theta_i$, that ground the clauses $c_i$ such that $\mathit{head}(c_1\theta_1) = \mathit{head}(c_2\theta_2)$. They specify $\mathit{cpd}(c_1\theta_1)$ and $\mathit{cpd}(c_2\theta_2)$, but one needs $p(\mathit{head}(c_1\theta_1) \mid \mathit{body}(c_1) \cup \mathit{body}(c_2))$. The standard solution to obtain the densities required are so called *combining rules* (see e.g. [20]), functions which map finite sets $\{p(A \mid A_{i1}, \ldots, A_{in_i}) \mid i = 1, \ldots, m\}$ of conditional probability functions onto one *combined* conditional probability function $p(A \mid B_1, \ldots, B_k)$ with $\{B_1, \ldots, B_k\} \subset \bigcup_{i=1}^{m} \{A_{i1}, \ldots, A_{in_i}\}$. We assume that for each Bayesian predicate $p$ there is a corresponding combining rule $\mathit{cr}(p)$, such as noisy_or in the case of discrete random variables or a linear regression model in the case of Gaussian variables.

To summarize, a *Bayesian logic program* $B$ consists of a (finite) set of Bayesian clauses. To each Bayesian clause $c$ there is an associated conditional probability function $\mathit{cpd}(c)$, and for each Bayesian predicate $p$ there is exactly one associated combining rule $\mathit{cr}(p)$.

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 $\theta$, 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 least Herbrand model $\mathrm{LH}(B)$ consists of all *proper* random variables. It is defined as if $B$ would be a logical definite program (cf. [17]). It is the least fix point of the *immediate consequence* operator[3](cf. [17]) applied on the empty set. Now, to each node $x$ in $DG(B)$ the *combined* pdf 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 $body(c\theta) \subset \mathrm{LH}(B)$. Thus, the dependency graph encodes similar to Bayesian networks the following independency assumption:

> *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 3 renders $h(john)$ independent from $g(uta)$ given a joint state of $g(john), h(uta), h(peter), m(uta, john), f(peter, john)$. Using this assumption the following proposition holds:

**Proposition 1.** *Let $B$ be a Bayesian logic program. If $B$ fulfills (1) that* $\mathrm{LH}(B) \neq \emptyset$*, (2) that $DG(B)$ is acyclic in the usual graph theoretical sense, and (3) that each node in $DG(B)$ is influenced by a finite set of random variables then it specifies a unique probability density over* $\mathrm{LH}(B)$.

*proof sketch (For a detailed proof see [13].).* The least Herbrand $\mathrm{LH}(B)$ always exists, is unique and countable. Thus, $DG(B)$ uniquely exists, and due to condition (3) the combined pdf for each node of $DG(B)$ is computable. Furthermore, because of condition (1) a total order $\pi$ of $DG(B)$ exists, so that one can see $B$ together with $\pi$ as a stochastic process over $\mathrm{LH}(B)$. An inductions "along" $\pi$ together with condition 2 shows that the family of finite-dimensional distribution of the process is projective (cf. see [1]), i.e the jpdf over each finite subset $s \subseteq \mathrm{LH}(B)$ is uniquely defined and $\int_y p(s, x = y)\, dy = p(s)$. With that, the preconditions of *Kolmogorov's theorem* [1, page 307] hold, and it follows that $B$ given $\pi$ specifies a probability density function $p$ over $\mathrm{LH}(B)$. This proves the proposition because the total order $\pi$ used for the induction does not refer to any specific total order of $DG(B)$.

A program fulfilling conditions 1, 2 and 3 is called *well-defined* and we will consider such programs for the rest of the paper. One can think of Bayesian networks as a simple example of well-defined programs. Their graphically represented dependencies are encoded as a finite propositional Bayesian logic program as shown in Figure 2. A program encoding the intensional regularities in our genetic domain is given in Figure 3. Some interesting properties follow from the proof sketch.

[3] We assume that all clauses in a Bayesian logic program are range-restricted: all variables appearing in the conclusion part of a clause also appear in the condition part. This is a common restriction in computational logic, because then all facts entailed by the program are ground (cf. [17]).

```
m(uta,peter).
f(john, peter).
g(uta).
g(peter).
g(X)      | m(M,X), g(M), f(F,X), g(F).
h(uta)    | g(uta).
h(peter)  | g(peter).
h(X)      | g(X), m(M,X), h(M), f(F,X), h(F).
```

**Fig. 3.** A Bayesian logic program encoding the example in our genetic domain.

- We interpreted a Bayesian logic program as a stochastic process. This places them in a wider context of what Cowell et. al. call *highly structured stochastic systems* (HSSS, cf. [3]) because Bayesian logic programs represent discrete-time stochastic processes in a more flexible manner. Well-known probabilistic frameworks such as dynamic Bayesian networks, first order hidden Markov models or Kalman filters are special cases of them.
- Together with the unique semantics for pure discrete programs [12] it is clear that hybrid programs, i.e. programs over discrete and continuous variables have a unique semantics.

Moreover, the proof in [13] indicates the important *support network* concept. Support networks are a graphical representation of the finite-dimensional distribution (cf. [1]) and are needed for the formulation of the likelihood function (see below) as well as for answering probabilistic queries in Bayesian logic programs. The *support network* $N$ of a variable $x \in \mathrm{LH}(B)$ is defined as the induced subnetwork of $S = \{x\} \cup \{y \mid y \in \mathrm{LH}(B) \text{ and } y \text{ is influencing } x\}$. The support network of a finite set $\{x_1, \ldots, x_k\} \subseteq \mathrm{LH}(B)$ is the union of the networks of each single $x_i$. Because we consider well-defined Bayesian logic programs, each $x \in \mathrm{LH}(B)$ is influenced by a finite subset of $\mathrm{LH}(B)$. So, it is provable that the support network $N$ of a finite set $\{x_1, \ldots, x_k\} \subseteq \mathrm{LH}(B)$ of random variables is always a finite Bayesian network and computable in finite time. Because the support network $N$ models the finite-dimensional distribution specified by $S$, any interesting probabilistic density value over subsets of $S$ is specified by $N$. For the proofs and an effective inference procedure (together with an implementation using Prolog) we refer to [13].

## 4 Maximum Likelihood Estimation

So far, we have assumed that there is an expert who designs a Bayesian logic program. This is not always the case. Often, there is no-one possessing necessary expertise or knowledge. However, we often have access to data. We focus here on the classical *maximum likelihood estimation* (MLE) method to learn the parameters of the associated probability density functions of a given Bayesian logic program.

Let $B$ be a Bayesian logic program consisting of the Bayesian clauses $c_1, \ldots, c_n$, and let $\mathbf{D} = \{D_1, \ldots, D_m\}$ be a set of data cases. A data case $D_i \in \mathbf{D}$ is a partially observed joint state of some variables in $\mathrm{LH}(B)$. Examples of data cases are

$$\{m(peter, uta) = true, f(peter, john) = true, h(uta) = 165.98, h(peter) = 175.8\},$$
$$\{g(peter) =?, h(uta) = 165.98, h(peter) = 174.4, h(john) = 170\},$$
$$\{h(uta) = 167.9, h(john) =?\},$$

where '?' stands for an unobserved state. The parameters $\boldsymbol{\lambda}(c_i) = \{\lambda(c_i)_1, \ldots, \lambda(c_i)_{e_i}\}$, $e_i > 0$, affecting the associated pdfs $cpd(c_i)$ constitute the set $\boldsymbol{\lambda} = \bigcup_{i=1}^{n} \boldsymbol{\lambda}(c_i)$ and the version of $B$ where the parameters are set to $\boldsymbol{\lambda}$ is denoted by $B(\boldsymbol{\lambda})$[4]. Now, the likelihood $L(\mathbf{D}, \boldsymbol{\lambda})$ is the probability of the observed data $\mathbf{D}$ as a function of the unknown parameters $\boldsymbol{\lambda}$:

$$L(\mathbf{D}, \boldsymbol{\lambda}) := P_B(\mathbf{D} \mid \boldsymbol{\lambda}) = P_{B(\boldsymbol{\lambda})}(\mathbf{D}). \tag{3}$$

Thus, the search space $\mathcal{H}$ is spanned by the product space over the possible values of $\lambda(c_i)$ and we seek to find $\boldsymbol{\lambda}^* = \max_{\boldsymbol{\lambda} \in \mathcal{H}} P_{B(\boldsymbol{\lambda})}(\mathbf{D})$. Usually, $B$ specifies a density function over a (countably) infinite set of random variables and hence we cannot compute $P_{B(\boldsymbol{\lambda})}(\mathbf{D})$ by considering the whole dependency graph. But as we have argued at the end of the preceding section it is sufficient to consider the support network $N(\boldsymbol{\lambda})$ of the random variables occurring in $\mathbf{D}$ to compute $P_{B(\boldsymbol{\lambda})}(\mathbf{D})$. Thus, remembering that the logarithm is monotone we seek to find

$$\boldsymbol{\lambda}^* = \max_{\boldsymbol{\lambda} \in \mathcal{H}} \log P_{N(\boldsymbol{\lambda})}(\mathbf{D}). \tag{4}$$

In other words, we have expressed the original problem in terms of the parameter MLE problem of Bayesian networks. However, we need to be more careful. Some of the nodes in $N(\boldsymbol{\lambda})$ are hidden, that is, their values are not observed in $\mathbf{D}$[5]. Furthermore, it should be noted, that not only $L(\mathbf{D}, \boldsymbol{\lambda})$ but also $N(\boldsymbol{\lambda})$ itself depends on the data, i.e. the data cases determine the sufficient subnetwork of $DG(B)$ to calculate the likelihood. On the one hand, this may affect the generalization of the learned program, but on the other hand, this is a similar situation as for "unrolling" dynamic Bayesian networks [5] or recurrent neural networks [24].

Finally, our learning setting can be used for MLE of the parameters of intensional rules only. Assume that if we observe $h(john)$ then $h(john) = 172.06$ holds. In this case, it is problematic to estimate the ML parameters of $h(john)$. But, we can still estimate the ML parameters of `h(X) | g(X)` based on the support network of the data cases: the intensional rules together with the data

[4] As long as no ambiguities occur we will not distinguish between the parameters $\boldsymbol{\lambda}$ themselves and a particular instance of them.

[5] If all nodes are observed in each $D_i \in \mathbf{D}$, then simple counting is all that is needd for ML parameter estimation in Bayesian networks (see e.g. [9]).

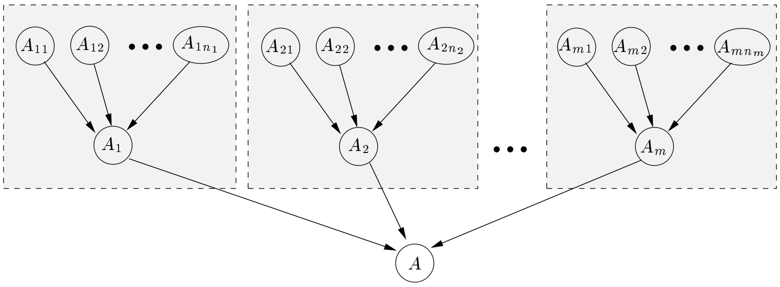


**Fig. 4.** The scheme of decomposable combining rules. Each rectangle corresponds to a ground instance of a Bayesian clause (cp. definition of a combining rule). The node $A$ is a deterministic node.

cases induce a Bayesian network over the variables of the data cases. This is not surprising if one sees our learning setting as a probabilistic extension of the ILP setting *learning from interpretation*. For a discussion on how this analogy can be used for learning intensional Bayesian clauses (not only the parameters of the associated densities) we refer to [14].

## 5 The Gradient

How can we maximize the likelihood? A classical method for finding a maximum of an evaluation function is *gradient ascent*, also known as *hill climbing*. Here, one computes the *gradient* vector $\nabla_{\boldsymbol{\lambda}}$ of partial derivatives with respect to the parameters of the pdfs at a given point $\boldsymbol{\lambda} \in \mathcal{H}$. Then it takes a small step in the direction of the gradient to the point $\boldsymbol{\lambda} + \alpha\nabla_{\boldsymbol{\lambda}}$ where $\alpha$ is the step-size parameter. The algorithm will converge to a local maximum for small enough $\alpha$. Thus, we have to compute the partial derivatives of $P_{N(\boldsymbol{\lambda})}(\mathbf{D})$ with respect to some particular parameter $\lambda(c)_t$ [6]. For the sake of simplicity we will assume *decomposable* combining rules[7]. Such rules can be expressed using a set of separate, deterministic nodes in the support network, as shown in Figure 4. Most combining rule commonly employed in Bayesian networks such as noisy_or or linear regression are decomposable (cp. [10]).

Decomposable combining rules imply that for each node $x \in N$ there exist at most one clause $c$ and a substitution $\theta$ s.t. $body(c\theta) \subset \mathrm{LH}(B)$ and $head(c\theta) = x$. Thus, while the same clause $c$ can induce more than one node in $N$, all of these nodes have identical local structure: the associated pdfs (and so the parameters)

[6] In the algorithm, this requires an additional step. We have to make sure that (1) each $cpd(c)$ maps into $[0, 1]$, and (2) for each $u \in \mathrm{dom}(head(c))$ and for each $\mathbf{u} \in \mathrm{dom}(body(c))$ : $\int_{-\infty}^{+\infty} cpd(c)(u, \mathbf{u})du = 1$. This can be done by renormalizing $\nabla_{\boldsymbol{\lambda}}$ to the constrained surface before taking a step in the direction of $\nabla_{\boldsymbol{\lambda}}$.

[7] In the case of more general combining rules the partial derivatives of a inner function has to be computed. This may be difficult or even not possible (in a close form).

have to be identical, i.e. $\forall$ *subst.* $\theta : cpd(c\theta) = cpd(c)$. As an example consider the clause defining `h(X)` and the nodes `h(uta)`, `h(peter)` and `h(john)`. This is the same situation as for dynamic Bayesian networks where the parameters that encode the stochastic model of state evolution appear many times in the network.

In the following we will adapt a solution based on the chain rule of differentiation given in [2] for dynamic Bayesian networks. For simplicity, we fix the current instantiation of the parameters $\boldsymbol{\lambda}$ and, hence, we write $B$ and $N(\mathbf{D})$. Applying the chain rule on (4) yields

$$\frac{\partial \log P_N(\mathbf{D})}{\partial \lambda(c)_t} = \sum_{\substack{\text{subst. } \theta \text{ with} \\ support(c\theta)}} \frac{\partial \log P_N(\mathbf{D})}{\partial \lambda(c\theta)_t} \tag{5}$$

where $\theta$ refers to grounding substitutions and $support(c\theta)$ is true iff $\{head(c\theta)\} \cup body(c_i\theta) \subset N$. Assuming that the data cases $D_l \in \mathbf{D}$ are independently sampled from the same distribution we can separate the contribution of the different data cases to the partial derivative of a single ground instance $c\theta$ resulting in[8]:

$$\frac{\partial \log P_N(\mathbf{D})}{\partial \lambda(c\theta)_t} = \sum_{l=1}^{m} \frac{\partial \log P_N(D_l)}{\partial \lambda(c\theta)_t} =$$
$$\sum_{l=1}^{m} \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} \frac{p_N(head(c\theta) = u, body(c\theta) = \mathbf{u} \mid D_l)}{cpd(c\theta)(u, \mathbf{u})} \times \frac{\partial cpd(c\theta)(u, \mathbf{u})}{\partial \lambda(c\theta)_t}\, du d\mathbf{u} \tag{6}$$

where $u \in \text{dom}(head(c\theta)$, $\mathbf{u} \in \text{dom}(body(c\theta)$. The term $p_N(u, \mathbf{u} \mid D_l)$ cannot be exactly calculated for all kinds of distributions. If not, stochastic simulation techniques such as Markov chain Monte Carlo methods (see e.g. [3, Appendix B]) should help. Another often used solution is to restrict the types of the random variable. Most continuous Bayesian networks have used Gaussian distributions for the density functions (e.g. *conditional Gaussian* distributions [3]). This can be done with Bayesian logic programs, too, so that the solution of the integrand in Equation (6) has a closed form which can be adapted from [2], and an inference engine for Bayesian logic programs can be used to get an exact solution. But still in general, the integrals in Equation (6) are intractable. Here again, stochastic simulation algorithms may solve the problem. We finally would like to state the equations of the partial derivatives for pure discrete programs. Doing the same steps as before (Equations (5), (6)) and noting that the densities are now distributions parameterized by their entries yields:

$$\frac{\partial \log P_N(\mathbf{D})}{\partial cpd(c)_{jk}} = \sum_{\substack{\text{subst. } \theta \text{ with} \\ support(c\theta)}} \sum_{l=1}^{m} \frac{P_N(head(c\theta) = u_j, body(c\theta) = \mathbf{u}_k \mid D_l)}{\partial cpd(c\theta)_{jk}} \tag{7}$$

where $u_i \in \text{dom}(head(c))$, $\mathbf{u}_j \in \text{dom}(body(c))$ and $i, j$ refer to the corresponding entries in $cpd(c)$ and $cpd(c\theta)$. With this, it is not difficult to adapt the equations

[8] Due to space restrictions we leave the derivation of the equation out. It is basically the derivation of equation (10) in [2] adapted to our notation.

**Table 1.** A simplified skeleton of the algorithm for *adaptive Bayesian logic programs.*.

```
function BASIC-ABLP(B, D) returns a modified Bayesian logic program
    inputs: B, a Bayesian logic program; associated pdfs are parameterized by λ
            D, a finite set of data cases

λ ←INITIALPARAMETERS
N ←SUPPORTNETWORK(B, D)
repeat until Δλ ≈ 0
    Δλ ← 0
    set pdfs of N according to λ
    for each D_l ∈ D
        set the evidence in N from D_l
        for each clause c ∈ B
            for each ground instance cθ s.t. {head(cθ)} ∪ body(cθ) ⊂ N
                for each single parameter λ(cθ)_t
                    Δλ(c)_t ← Δλ(c)_t + (∂ log P_N(D_l)/∂λ(cθ)_t)
    Δλ ←PROJECTIONONTOCONSTRAINTSURFACE(Δλ)
    λ ← λ + α · Δλ
return B
```

for hybrid Bayesian logic programs. A simplified skeleton of a gradient-based algorithm is shown in Table 1.

## 6 Related Work

To some extent, Bayesian logic programs are related to the BUGS language [8] which aims at carrying out Bayesian inference using Gibbs sampling. It uses concepts of imperative programming languages such as for-loops to model regularities in probabilistic models. Therefore, the relation between Bayesian logic programs and BUGS is akin to the general relation between logical and imperative languages. This holds in particular for relational domains such as that used in this paper: family relationships. Without the notion of objects and relations among objects family trees are hard to represent. Furthermore, a single BUGS program specifies a probability density over a finite set of random variables, whereas a Bayesian logic program can represent a distribution over an infinite set of random variables.

There is work on parameter estimation within "first order probabilistic logics" which do not rely on Bayesian networks. Cussens [4] investigates EM methods to estimate the parameters of stochastic logic programs [19]. Sato et al. [23] have shown that there is an efficient method for EM learning of PRISM programs.

Learning within Bayesian networks is well-investigated in the Uncertainty in AI community, see e.g. [9]. Binder et. al. [2] whose approach we have adapted present results for a gradient-based method. But so far, there has not been much work on ML parameter estimation within first order extensions of Bayesian networks. Koller and Pfeffer [16] adapt the EM algorithm for probabilistic logic programs [20], a framework which in contrast to Bayesian logic programs sees ground atoms as states of random variables. Although the framework seems to theoretically allow for continuous random variables there exists no (practical) query-answering procedure for this case; to the best of our knowledge Ngo and Haddawy [20] give only a procedure for variables having finite domains. Furthermore, Koller and Pfeffer's approach utilizes support networks, too, but requires the intersection of the support networks of the data cases to be empty. This could be in our opinion in some cases too restrictive, e.g. in the case of dynamic Bayesian networks. However, if the data cases fulfill the property then inference is faster. Friedman et al. [7] concentrate on learning within probabilistic relational models, a framework that combines entity/relationship models with Bayesian networks. They adapt the EM algorithm to estimate the parameters and to learn the structure based on techniques known from Bayesian network learning. So, they consider a more general problem setting than we do, but on the other hand the entity/relationship model lacks the concept of functors. Thus, they are limited to finite sets of entities and relations. For a more detailed discussion of the relations of Bayesian logic programs to other first order extensions of Bayesian networks such as probabilistic logic programs [20], relational Bayesian networks [11] and probabilistic relational models [15] we refer to [12, 13].

Therefore, the related work on first order extensions of Bayesian networks mainly differs in two points from ours: (1) The underlying (logical) frameworks lack important knowledge representational features which Bayesian logic programs have. (2) They adapt the EM algorithm which is particularly easy to implement. However, there are problematic issues both regarding speed of convergence as well as convergence towards a local (sub-optimal) maximum of the likelihood function. Different accelerations based on the gradient are discussed in [18]. Also, the EM algorithm is difficult to apply in the case of general probability density functions because it relies on computing the sufficient statistics (cf. [9]).

## 7 Experimental Prospects

The experimental results of [16] and [2] can be summarized as follows: (1) the support network is a good approximation of the entire likelihood, (2) equality constraints over parameters speed up learning, and (3) gradient-based methods are promising. Therefore, we prove the basic principle of our approach by testing the hill-climbing algorithm on a simple model of our genetic domain. We generated 100 data cases from a version of the program in Figure 3 where the genetical information expressed by $g$ is omitted. It describes the family tree of 12 person. The associated probability functions are $\mathit{cpd}(m(M,X))(\mathit{true}) = \mathit{cpd}(f(F,X))(\mathit{true}) = 1.0$, $\mathit{cpd}(h(X)) = \mathcal{N}(165, 20)$ and the one in Table 2, where $\mathcal{N}(165, 20)$ denotes a normal density with mean 165 and variance 20. The learning task was to estimate $a$ in the function of Table 2 and the mean $b$ of

| $m(M,X)$ | $f(F,X)$ | $cpd(c)(h(X) \mid h(M), h(F))$ |
|---|---|---|
| *true* | *true* | $\mathcal{N}(0.0 + 0.5 \cdot h(M) + 0.5 \cdot h(F), 20)$ |
| *true* | *false* | $\mathcal{N}(165, 20)$ |
| *false* | *true* | $\mathcal{N}(165, 20)$ |
| *false* | *false* | $\mathcal{N}(165, 20)$ |

**Table 2.** The probability density function used in our experiments. Only in the case *true*, *true* the heights of the parents a taken into account as weighted sum. The constant addend in the mean of the first normal densities is denoted as $a$. i.e. $a = 0.0$.

$cpd(h(X))$ starting with $a = 165.0$ and $b = 0.0$. After 13 iterations the estimated parameters were $a = -0.1856$ and $b = 164.7919$ using a step-size of 1.0. The implementation obviously suffers from the well-known dependency on the chosen initial parameters and fixed step-size. In the future we will investigate more advanced gradient-based methods like e.g. Conjugate-Gradient. We also conducted experiments with learning the weights of the sum in the function of Table 2, i.e. 0.5 and 0.5. Here, the algorithm converges to weights almost summing to 1.0 which are local minima w.r.t. our data generating model and the likelihood.

# 8 Conclusions

We made two contributions. First, we have introduced continuous Bayesian logic programs. We have argued that the basic query-answering procedure for discrete programs is still applicable: The ability to represent both intensional regularities between the variables as well as continuous random variables reduces the size of many modelled domains. Second, we have addressed the question "where do the numbers come from?" by showing how to compute the gradient of the likelihood based on ideas known for (dynamic) Bayesian networks. The intensional representation of Bayesian logic programs, i.e. their compact representation should speed up learning and provide good generalization.

In the future, we will perform a detailed comparison of our learning approach with the EM algorithm. Accelerations of the EM algorithm based on the gradient are interesting. Our ultimate goal is learning the structure. We are currently (see [14]) looking for combinations of techniques known from Inductive Logic Programming, such as refinement operators, with techniques like scoring functions of the Bayesian networks. Just like ML parameter estimation is a basic technique for structural learning of Bayesian networks, it seems to be a basic technique for structural learning of Bayesian logic programs.

**Acknowledgments** The authors would like to thank Stefan Kramer and Manfred Jaeger for helpful discussions on the ideas of the paper. Also, thanks to the anonymous reviewers for their helpful comments.

## References


1. Heinz Bauer. *Wahrtscheinlichkeitstheorie.* Walter de Gruyter, Berlin, New York, 4. edition, 1991.
2. J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, pages 213–244, 1997.
3. R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic networks and expert systems.* Springer-Verlag New York, Inc., 1999.
4. J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 2001. to appear.
5. T. Dean and K. Kanazawa. Probabilistic temporal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-1988)*, 1988.
6. D. S. Falconer. *Introduction to quantitative genetics.* Longman Inc., New York, 2. edition, 1981.
7. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI-1999)*, 1999.
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,, 1995.
10. D. Heckerman and J. Breese. Causal Independence for Probability Assessment and Inference Using Bayesian Networks. Technical Report MSR-TR-94-08, Microsoft Research, 1994.
11. M. Jaeger. Relational Bayesian networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-1997)*, 1997.
12. 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/`.
13. K. Kersting and L. De Raedt. Bayesian logic programs. Technical Report 151, University of Freiburg, Institute for Computer Science, April 2001.
14. K. Kersting and L. De Raedt. Towards Combining Inductive Logic Programming and Bayesian Networks. In this volume, 2001.
15. D. Koller. Probabilistic relational models. In *Proceedings of Ninth International Workshop on Inductive Logic Programming (ILP-1999)*, 1999.
16. D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *Proceedings of the Fifteenth Joint Conference on Artificial Intelligence (IJCAI-1997)*, 1997.
17. J. W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, 2. edition, 1989.
18. G. J. McKachlan and T. Krishnan. *The EM Algorithm and Extensions.* John Eiley & Sons, Inc., 1997.
19. S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming.* IOS Press, 1996.
20. L. Ngo and P. Haddawy. Answering queries form context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1997.
21. J. Pearl. *Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 2. edition, 1991.
22. D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.

23. 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.
24. R. J. Williams and D. Zipser. Gradient-Based Learning Algorithms for Recurrent Networks and Their Computatinal Complexity. In *Back-propagation: Theory, Architectures and Applications*. Hillsdale, NJ: Erlbaum, 1995.