Linköping Electronic Articles in
Computer and Information Science
Vol. 5(2000): nr 34

Bayesian Logic Programs
(extended abstract)

Kristian Kersting and Luc De Raedt

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis/2000/034/

Published on December 21, 2000, 2000 by
Linköping University Electronic Press
581 83 Linköping, Sweden

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

Typeset by the author using LaTeX
Formatted using étendu *style*

Recommended citation:
<Author>. <Title>. Linköping Electronic Articles in
Computer and Information Science, Vol. 5(2000): nr 34.
http://www.ep.liu.se/ea/cis/2000/034/. *December 21, 2000, 2000.*

This URL will also contain a link to the author's home page.

Abstract

Bayesian networks provide an elegant formalism for representing and reasoning about uncertainty. They are a probabilistic extension of propositional logic and, hence, inherit some of the limitations of propositional logic, such as the difficulties to represent objects and relations. The main contribution of this extended abstract is to introduce a new approach, called Bayesian logic programs, to overcome the limitations. It combines Bayesian networks with definite clause logic, i.e. "pure" Prolog, by establishing a one-to-one mapping between ground atoms and random variables. Thus, Bayesian logic programs combine the advantages of definite clause logic and Bayesian networks. This includes the separation of quantitative and qualitative aspects of the world. Furthermore, Bayesian logic programs generalize both Bayesian networks as well as logic programs, many ideas developed in both areas can be adapted.

Authors' address

Machine Learning Lab
Institute of Computer Science
Albert-Ludwigs University
Georges-Koehler-Allee 079
D-79085 Freiburg, Germany

1 Introduction

A Bayesian network [17] specifies a joint probability distribution[1] $\mathbf{P}(X_1, \ldots, X_n)$ over a fixed set $X_1, \ldots, X_n$ of random variables. It consists of two components: (1) a directed acyclic graph indicating local influences between the random variables (which are the nodes of the graph) and (2) a set of conditional probability distributions $\mathbf{P}(X_i \mid Pa(X_i))$ quantifying for each X_i the local influences of its parents $Pa(X_i)$, the direct predecessors of X_i. As an illustration consider a variant of the famous example due to Judea Pearl about burglary alarms at home which is shown in Figure 1. It states e.g. that the random variable *AlarmPeter* is influenced by *Burglary* and *Earthquake* through an associated conditional probability distribution $\mathbf{P}(\mathit{AlarmPeter} \mid \mathit{Burglary}, \mathit{Earthquake})$. Another interpretation of Bayesian networks is that they represent a set of conditional independency statements, of which the simplest ones are: each random variable is conditional independent of any predecessor given a joint state of its parents. Russell and Norvig [20, p. 464] write that Bayesian networks "are a natural way to represent conditional independence information. The links between nodes represent qualitative aspects of the domain, and the conditional probability tables represent the quantitative aspects." Thus, the strict separation of qualitative and quantitative aspects is an important feature of Bayesian networks.

Bayesian networks provide an elegant probabilistic extension of propositional logic. However, the limitations of propositional logic, which Bayesian networks inherit, are well-known [15, 8, 3, 11]: they have a rigid structure and have problems representing a variable number of objects or relations among objects. E.g. we have to rebuild our example Bayesian network if we want to model a second alarm system of a house in the neighbourhood of Peter sharing only the *Earthquake* and *Burglary* nodes (see Figure 1). This in turn exemplifies another aspect of the propositional character: we have to store the whole net, even if we focus for some queries only on one of the alarm systems. In this context, it is no surprise that various researchers have proposed first order extensions of Bayesian networks, e.g. [19, 15, 8, 11]. Many of these techniques employ the notion of *knowledge-based model construction* (KBMC) [6], where first-order rules with associated uncertainty parameters are used to generate Bayesian networks for particular queries. This is especially useful in domains where the number of relevant random variables depends on the specific problem. E.g. when reasoning about the probability that a person inherits a genetic disease, the relevant random variables will depend crucially on the available knowledge and structure of the family of that person.

We tried to identify an intuitive and straightforward formalism of first order extension of Baysian networks, that is as close as possible to both Bayesian nets and to some well-founded first order logic knowledge representation mechanism, in our case definite clause logic, i.e. "pure" Prolog programs. Any formalism designed according to this principle should be easy to understand and use by researchers in both communities.

The paper is laid out as follows. Section 2 introduces the representation language of Bayesian logic programs and Section 3 gives the declarative semantics. Section 4 briefly discusses related work. The last Section

---

*This extended abstract summarizes one of the presentations at the Seventeenth Machine Intelligence Workshop (MI-17). It has been submitted to the special issue so that the special issue provides a genuine overview of MI-17.

[1] We denote a single probability with P and a distribution with $\mathbf{P}$

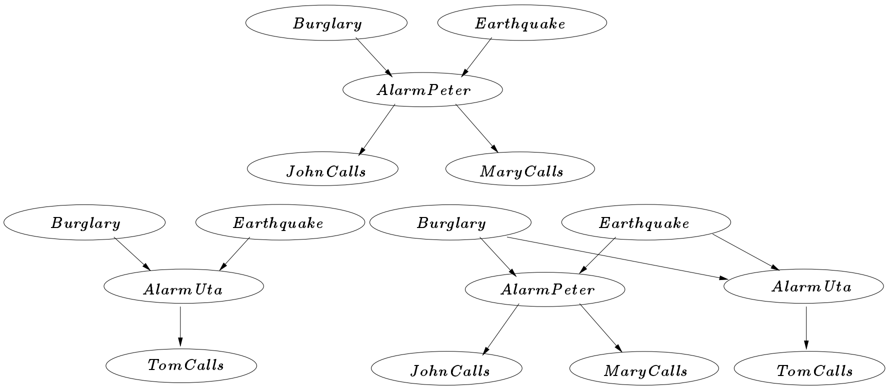

Figure 1: Three Bayesian networks (associated conditional probability distributions omitted) about burglary alarms at the homes of Peter and Uta (cf. [20]). The upper one is the classical one. The lower left one is the second alarm system where all occurring conditional probability distributions are the same as in the upper one. The lower right one models both alarm systems.

concludes the paper and states future work. Here, we mainly summarize the underlying ideas of Bayesian logic programs. A detailed discussion of Bayesian logic programs including a query-answering procedure, a comparison to other first order extensions of Bayesian nets like [15, 8, 4] and an interpretation of Bayesian logic programs using a Prolog meta interpreter can be found in Kersting's thesis [9].

We assume some familiarity with logic programming or Prolog (see e.g. [22]) as well as with Bayesian networks (see e.g. [20]).

2 Representation language

A Bayesian logic program consists of two components. The first component is the logical one which consists of a set of Bayesian clauses (cf. below). It captures the qualitative structure of the domain and is based on definite clause logic, i.e. "pure" Prolog. The second component is the quantitative one which encodes the quantitative aspects of the domain. It employs - as in Bayesian networks - the notions of conditional probability distribution and combining rule (cf. below).

A *Bayesian predicate* is a predicate r, called the corresponding logical predicate, to which a discrete domain D_r is associated. Notice that r plays the role of an identifier, i.e. given r the associated domain D_r and so the intended Bayesian predicate is unique. Analogously, we speak about *Bayesian (ground) atoms* and their corresponding logical counterparts. A Bayesian atom a over r shares its domain with r, i.e. $D_a := D_r$. We define a *Bayesian (definite) clause* as an expression of the form $A \mid A_1, \ldots, A_n$ where the $A, A_1, \ldots, A_n$ are Bayesian atoms and all variables are (implicitly) universally quantified. When writing down Bayesian definite clauses we will closely follow Prolog notation (with the exception that Prolog's "`:-`" is replaced by "`|`"). So, variables start with a capital, constant and functor symbols start with a lowercase. The main difference between Bayesian and logical clauses is that Bayesian atoms represent sets of similar random variables. More precisely, each Bayesian ground atom represents a random variable which takes exactly one value of its associated (discrete) domain in any state of the world. Therefore, a logical predicate l is a special case of a Bayesian one with $D_l = \{true, false\}$. An example of a Bayesian definite clause inspired on [15] is

```
burglary(X) | neighbourhood(X).
```

where the domains are $D_{burglary} = \{yes, no\}$ and $D_{neighbourhood} = \{bad, average, good\}$. Roughly speaking, a Bayesian definite clause specifies that for each substitution θ (cf. [12]) that grounds the clause the random variable $A\theta$ depends on $A_1\theta, \ldots, A_n\theta$ if the latter ones are proper random variables. Proper random variables correspond to ground atoms that are true in the logical sense (cf. below). This ensures the existence of a unique probability distribution over the relevant random variables of our domain (see next Section). For instance, let $\theta = \{X \leftarrow james\}$, then the random variable *burglary*(*james*) depends on *neighbourhood*(*james*) if *neighbourhood*(*james*) follows from the Prolog program.

As for Bayesian networks there exists a conditional probability distribution associated to each Bayesian definite clause

neighbourhood(X)	*burglary*(X) *true*	*burglary*(X) *false*
bad	0.6	0.4
average	0.4	0.6
good	0.3	0.7

which specifies the knowledge about the conditional probability distribution $\mathbf{P}(A\theta \mid A_1\theta, \ldots, A_n\theta)$ for every ground instance θ of the clause. For instance, the above Bayesian definite clause and conditional probability distribution together imply that $P(burglary(james) = true \mid neighbourhood(james) = bad) = 0.6$. We use a naive representation as a table, because it is the simplest representation. We stress, however, that other representations are possible and known [1].

Each Bayesian predicate is defined by a set of definite Bayesian clauses, e.g.

```
alarm(X) | burglary(X).
alarm(X) | tornado(X).
```

But, if there are more than one different ground instance of a clause with the same Bayesian ground atom as head, we have multiple conditional probability distributions over the random variable represented by the Bayesian ground atom in the head. E.g. given the clauses for `alarm` and the Bayesian ground facts `burglary(john).` and `tornado(john).`, the random variable *alarm*(*james*) depends on both *burglary*(*james*) and *tornado*(*james*). However, the conditional probability distributions for *alarm* do not specify $\mathbf{P}(alarm(james) \mid burglary(james), tornado(james))$. The standard solution to obtain the latter probability distribution from the associated conditional probability distributions is to use a so called *combining rule*. A combining rule is any algorithm which

1. maps every finite set of conditional probability distributions $\{\mathbf{P}(A \mid A_{i1}, \ldots, A_{in_i}) \mid 1 \leq i \leq m,\ n_i \geq 0\}$, $m \geq 1$, onto one conditional probability distribution, called combined conditional probability distribution, $\mathbf{P}(A \mid B_1, \ldots, B_n)$ with $\{B_1, \ldots, B_n\} \subseteq \bigcup_{i=1}^{m} A_{i1}, \ldots, A_{in_i}$ and

2. whose output is empty iff its input is empty.

Our definition of a combining rule is basically a reformulation of the definition given in [15] at the level of conditional probability distributions and differs mainly in the restriction of the input set to be finite. We make this assumption in order to keep the framework simple.

A *Bayesian logic program* B consists of a set of Bayesian definite clauses. Exactly one conditional probability distribution is associated to each Baysian clause in B and exactly one combining rule is associated to each Bayesian predicate in B.

3 Declarative semantics

The general idea is that each Bayesian logic program specifies in principle a (possible infinite) Bayesian network. The graphical structure of the Bayesian network follows from the semantics of the corresponding logic program (cf. below), whereas the quantitative aspects are encoded in the combined conditional probability distributions.

The meaning of definite logic programs has been well-studied (see e.g. [12]) and is given by its least Herbrand model. The least Herbrand model $\mathrm{LH}(L)$ of a program L contains the set of all ground atoms that are logically entailed by the program, it represents the intended meaning of the program. The *corresponding logic program* $\widetilde{B}$ is constructed from a Bayesian logic program B by replacing each Bayesian atom with its corresponding logical atom and the symbol "`|`" with "`:-`" in each clause belonging to B. The role of the *corresponding logic program* $\widetilde{B}$ is to specify the set of proper random variables and their local dependencies. More precisely, the proper random variables specified by B are the random variables corresponding to the members of the least Herbrand model $\mathrm{LH}(\widetilde{B})$. Because of the one-to-one mapping between a random variable and its corresponding ground atom we will also speak about the least Herbrand model $\mathrm{LH}(B)$ of the Bayesian logic program B. Analogously, we transfer any concept concerning logic programs to Bayesian logic programs. E.g. we talk about the Herbrand base of B which is the set of all random variables which can be formed using B (cf. [12]). By varying the "evidence" (some of the Bayesian ground facts) one also modifies the set of random variables. For example, if one takes as a Bayesian logic program the union of all Bayesian clauses written above together with *neighbourhood(james)* then $\mathrm{LH}(B)$ consists of *neighbourhood(james)*, *burglary(james)* and *alarm(james)*. Notice that the least Herbrand model can be infinite when the logic program contains structured terms. For simplicity, we will assume that all clauses in a Bayesian logic program are range-restricted. This means that all variables appearing in the conclusion part of a clause also appear in the condition part. This is a common restriction in computational logic. If a program contains only range-restricted clauses, all facts entailed by the program are ground.

Given two random variables A and $C \in \mathrm{LH}(B)$, we write that A *is directly influenced by* C if and only if there is a clause $A' \mid C_1, \ldots, C_n$ in B and a substitution θ that grounds the clause such that $A = A'\theta$ and $C = C_j\theta$ for some j, $1 \leq j \leq n$, and for all $C_i\theta \in \mathrm{LH}(B)$, $1 \leq i \leq n$. The relation *influenced by* is then the recursive closure of the relation *directly influenced by*. Roughly speaking, a (Bayesian) ground atom A influences C whenever there exists a proof for C that employs A. For instance, *alarm(james)* is influenced by *neighbourhood(james)* and directly influenced by *burglary(james)*. Using the *influenced by* relation we can now state a conditional independency assumption (akin to the one of Bayesian networks): Let $\mathbf{A}$ be the set of all random variables that directly influence a variable A. Then A is conditionally independent of any random variable C influencing A given a joint state of $\mathbf{A}$ i.e. $\mathbf{P}(A \mid \mathbf{A}, C) = \mathbf{P}(A \mid \mathbf{A})$.

So far, we have ignored one important requirement: the relation *influenced by* should be acyclic in order to obtain a well-defined Bayesian net. The network can only be cyclic when there exist a random variable or rather a ground atom A that is influenced by itself. When there exists such an A, executing the query ?- A (using Prolog) is also problematic (the SLD tree (cf. [12]) of the query will be infinite and the query may not terminate). Thus in such cases the logical component of the Bayesian logic program is itself problematic. Additional considerations lead to the following proposition.

Proposition 1 (the proof can be found in [9]). *Let B be a Bayesian logic program. If*

1. *the influenced by relation over* $\mathrm{LH}(B)$ *is acyclic and*
2. *each random variable in* $\mathrm{LH}(B)$ *is only influenced by a finite set of random variables,*

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

The conditions still allow infinite least Herbrand models but account for Bayesian networks: they are directed, acyclic graphs whose nodes have only a finite set of direct predecessors. Let us have a look at a program which violates conditions 1 and 2:

```
s(a).
r(X)    | r(f(X)).
r(f(X)) | s(f(X)).
s(f(X)) | s(X).
```

The random variable $r(a)$ is influenced (not directly) by $r(f(a))$, $r(f(f((a)))$,... though it has a finite proof. We assume that a Bayesian logic program is unproblematic in this respect[2]: a Bayesian logic program is called *well-defined* if it fulfills condition 1 and 2.

4 Related work

Bayesian logic progams are motivated by the formalisms discussed in [5, 15, 8, 4, 11]. They are most closely related to Ngo and Haddawy's KBMC framework of probabilistic logic programs [15] even though Ngo and Haddawy interpret a ground atom as a state of a random variable and augment clauses with a single probability value instead of with a conditional probability distribution. As follows from a detailed discussion in [9] both frameworks essentially have the same expressiveness. Due to the close relation with probabilitistic logic programs, Bayesian logic programs also extend Poole's framework of probabilistic Horn abduction [19] because probabilistic logic programs do so, cf. see [15]. The idea of associating conditional probability distributions to clauses is also proposed by Fabian and Lambert [3], though they see a ground atom as a random variable over $\{true, false\}$ and give a quite different declarative semantics. E.g. they do not have the concept of a combining rule but instead use a kind of backtracking mechanism

[2] This is a reasonable assumption if the Bayesian logic program has been written by anyone familiar with Prolog.

to cope with the situation where one atom can be proved in different ways. Bayesian logic programs are also related to Jaeger's relational Bayesian networks [8] and Koller et. al.'s probabilistic relational models [11, 4]. The former ones are Bayesian networks over set-valued random variables where the nodes correspond to predicate symbols and the associated domains are interpretations. The latter framework of probabilistic relational models combines entity/relationship models with Bayesian networks by interpreting attributes of entities as random variables. A detailed comparison in [9] shows that Bayesian logic programs can serve as the common kernel of these two frameworks and that of probabilistic logic programs.

Many other proposals that combine first-order logic and probability theory but do not extend Bayesian networks are known. This includes the work of Ng and Subrahmanian [14] who introduced a probabilistic characterization of logic programming and the work of Sato [21]. Parson's article [16] gives a general survey. Stochastic logic programs [13, 2] introduced by Muggleton are another interesting approach. They are logic programs consisting of labeled and unlabeled clauses. In contrast to Bayesian logic programs, probabilities in stochastic logic programs are defined directly on the proofs of atomic formulas.

5 Conclusions

We have introduced the representation language and the declarative semantics of Bayesian logic programs, a new framework for combining Bayesian networks with definite clause logic, in our case "pure" Prolog. The main ideas of Bayesian logic programs are (1) to establish a one-to-one mapping between ground atoms and (discrete) random variables and (2) to consider only true ground atoms in order to ensure a unique probability distribution over the random variables. The least Herbrand model of a Bayesian logic program together with its direct influence relation is interpretable as a (possible infinite) Bayesian network where the parents of a random variable v are (Bayesian) ground atoms directly influencing v. Except for possibly infinite (in the above sense) Bayesian logic programs, any algorithm solving the inference problem for Bayesian networks can be applied. Therefore, Bayesian logic programs inherit the advantages of both Baysian networks and definite clause logic, including the strict separation of qualitative and quantitative aspects. Further topics concerning Bayesian logic programs like a query-answering procedure, a comparison to other existing first order extensions of Bayesian networks [15, 8, 4] and an interpretation of Bayesian logic programs using a Prolog meta interpreter can be found in [9].

In the future Bayesian logic programs will be applied to some real-world problems. Also the issues of learning Bayesian logic programs and Bayesian logic programs over continuous random variables will be investigated. We believe the elaborated features (like the strict separation of qualitative and quantitative aspects) of Bayesian logic programs will be advantageous. Some suggestions concerning learning can be founded in [10]. Also, the investigation of more efficient inference algorithms following the idea of structured inference [18] could be adapted. Furthermore, exploring the relations between Bayesian logic programs to other approaches not extending Bayesian networks (such as stochastic logic programs) is interesting. This question seems to be related to the relations between probabilistic structures inducing probabilities on possible worlds and inducing probabilities on the domain (due to Halpern [7]).

Acknowledgements

We would like to thank James Cussens, Peter Flach, Manfred Jaeger, Daphne Koller, Stefan Kramer and Stephen Muggleton for discussions and encouragement on Bayesian logic programs.

References

[1] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Contect-specific independence in Bayesian networks. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI-1996)*, 1996.

[2] J. Cussens. Loginear models for first-order probabilistic reasoning. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-1999)*, 1999.

[3] I. Fabian and D. A. Lambert. First-order Bayesian reasoning. In G. Antoniou and S. Slaney, editors, *Proceedings of 11th Australian Joint Conference on Artificial Intelligence*, number 1502 in LNAI. Springer, 1998.

[4] 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.

[5] P. Haddawy. Generating Bayesian networks from probabilistic logic knowledge bases. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-1994)*, 1994.

[6] P. Haddawy. An Overview of Some Recent Developments on Bayesian Problem-Solving Techniques. *AI Magazine - Special Issue on Uncertainty in AI*, 20(2):11–29, 1999.

[7] J. Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1989.

[8] M. Jaeger. Relational Bayesian networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-1997)*, 1997.

[9] K. Kersting. Bayes'sche-logische Programme. Master's thesis, Albert-Ludwigs-University, Freiburg, Germany, 2000.

[10] K. Kersting, L. De Raedt, and S. Kramer. Interpreting Bayesian Logic Programs. In L. Getoor and D. Jensen, editors, *Working Notes of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data (SRL)*, 2000.

[11] D. Koller. Probabilistic relational models. In S. Dzeroski and P. Flach, editors, *Proceedings of Ninth International Workschop on Inductive Logic Programming (ILP-1999)*. Springer, 1999.

[12] J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 2. edition, 1989.

[13] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.

[14] R. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[15] L. Ngo and P. Haddawy. Answering queries form context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1997.

[16] S. Parsons. Current approaches to handling imperfect information in data and knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):353–372, June 1996.

[17] J. Pearl. *Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2. edition, 1991.

[18] A. J. Pfeffer. *Probabilistic Reasoning for Complex Systems*. PhD thesis, Stanford University, 1999.

[19] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.

[20] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., 1995.

[21] T. Sato. A Statistical Learning Method for Logic Programs with Distribution Semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP-1995)*, pages pp. 715 – 729, 1995.

[22] L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. The MIT Press, 1986.