Extending ProbLog with Continuous Distributions

Bernd Gutmann[1], Manfred Jaeger[2], and Luc De Raedt[1]

[1] Department of Computer Science, Katholieke Universiteit Leuven, Belgium
{bernd.gutmann, luc.deraedt}@cs.kuleuven.be
[2] Department of Computer Science, Aalborg University, Denmark
jaeger@cs.aau.dk

Abstract. ProbLog is a recently introduced probabilistic extension of Prolog. The key contribution of this paper is that we extend ProbLog with abilities to specify continuous distributions and that we show how ProbLog's exact inference mechanism can be modified to cope with such distributions. The resulting inference engine combines an interval calculus with a dynamic discretization algorithm into an effective solver.

1 Introduction

Continuous distributions are needed in many applications for building a natural model. Probabilistic logic programming languages, such as ProbLog and CP-Logic [1], have, so far, largely focused on modeling discrete distributions and typically perform exact inference. The PRISM [2] system provides primitives for Gaussian distributions but requires the exclusive explanation property which complicates modeling. On the other hand, many of the functional probabilistic programming languages, such as BLOG [3] and Church [4], can cope with continuous distributions but only perform approximate inference by a Markov Chain Monte Carlo approach. Typical statistical relational learning systems such as Markov Logic and Bayesian Logic Programs have also been extended with continuous distributions. The key contribution of this paper is a simple probabilistic extension of Prolog based on the distribution semantics with both discrete and continuous distributions. This is realized by introducing a novel type of probabilistic fact where arguments of the fact can be distributed according to a continuous distribution. Queries can then be posed about the probability that the resulting arguments fall into specific intervals. We introduce the semantics of using continuous distributions in combination with comparison operations and show how ProbLog's inference mechanism, based on Binary Decision Diagrams (BDDs) [5], can be extended to cope with these distributions. The resulting language is called Hybrid ProbLog.

Similarly to Hybrid ProbLog, Hybrid Markov Logic Networks (HMLNs) [6] aim at integrating Boolean and numerical random variables in a probabilistic-logic modeling framework. The kind of modeling supported by HMLNs is quite different in nature from the kind of modeling for which Hybrid ProbLog is designed. In an HMLN, one defines equations that function as soft constraints for

relationships among numerical and logical variables. For example, one could express that the temperature on day d is typically around 20° Celsius using the weighted equality w $\texttt{temperature}(d) = 20$, where larger weights w lead to a larger penalty for deviations of *temperature*(d) from 20. All weighted formulae containing *temperature*(d), together, implicitly define a probability distribution for the random variable *temperature*(d) due to HMLN semantics. However, one cannot directly specify this distribution to be Gaussian with, for example, mean 20 and standard deviation 5. No exact inference methods have been developed for HMLNs.

It is also instructive to compare Hybrid ProbLog with Hybrid Bayesian Networks [7]. Apart from one being a logical-relational, and the other a purely propositional framework, there is also a key difference in the interaction between continuous and discrete random variables permitted. In Hybrid Bayesian Networks, the distributions of continuous variables (usually Gaussian) are typically conditioned on discrete variables, but continuous variables cannot be parents of discrete ones. In Hybrid ProbLog this order is reversed: continuous variables are at the roots of the directed model, and the discrete (Boolean) variables are conditioned on the continuous ones. Thus, Hybrid ProbLog provides modeling capabilities and exact inference procedures that, for the propositional case, are in some sense complementary to Hybrid Bayesian networks.

This paper has three main contributions. (1) An extension of ProbLog with continuous distributions, (2) formal study of its semantics, and (3) an efficient inference algorithm based on dynamic discretization.

The rest of this paper is organized as follows. Section 2 reviews basic concepts from ProbLog. Section 3 introduces the syntax and semantics of Hybrid ProbLog. Section 4 describes our exact inference algorithm. Before concluding, we evaluate the algorithm in Section 5.

2 ProbLog

ProbLog [8] is a recent probabilistic extension of Prolog. A ProbLog theory $T = F \cup \mathcal{BK}$ consists of a set of labeled facts $F = \{p_1 :: f_1, \cdots, p_n :: f_n\}$ and a set of definite clauses $\mathcal{BK}$ that express the background knowledge. The facts $p_j :: f_j$ in F are annotated with a probability p_j stating that $f_j\theta$ is true with probability p_j for all substitutions θ grounding f_j. The resulting facts $f_j\theta$ are called atomic choices and represent random variables; they are assumed to be mutually independent. It is not allowed to use a probabilistic fact in the heads of clauses in $\mathcal{BK}$. Let $\Theta = \{\theta_{j1}, \ldots \theta_{ji_j} | j = 1, \ldots n\}$ be a finite[3] set of possible substitutions for the variables in the probabilistic facts where i_j is the number of substitutions for fact j, then a ProbLog theory describes a probability distribution over Prolog programs $L \subseteq L_F$ where $L_F = F\Theta$ and $F\Theta$ denotes the set of all possible ground instances of facts in F:

$$P_P(L|F) := \prod_{f_j\theta_{jk} \in L} p_j \prod_{f_j\theta_{jk} \in L_F \setminus L} (1 - p_j) \ . \tag{1}$$

[3] Throughout the paper, we assume that $F\Theta$ is finite, but see [2] for the infinite case.

The *success probability* of a query q then is

$$P_s(q|T) := \sum_{\substack{L \subseteq L_F: \\ L \cup \mathcal{BK} \models q}} P(L|F) \ . \tag{2}$$

ProbLog also defines a probability distribution P_w over possible worlds, that is Herbrand interpretations. Each total choice $L \subseteq L_F$ can be extended to a possible world by computing the least Herbrand model of L. This possible world is assigned the probability $P_w = P(L|F)$. Thus a set of total choices represents an assignment of truth-values to all atomic choices.

3 Hybrid ProbLog

A Hybrid ProbLog theory $T = F \cup F^c \cup \mathcal{BK}$ is a ProbLog theory extended by a set of continuous probabilistic facts[4] of the form

$$F^c = \{(X_1, \phi_1) :: f_1^c, \cdots, (X_m, \phi_m) :: f_m^c\}$$

where X_i is a Prolog variable, appearing in the atom f_i^c and ϕ_i is a density function. The fact $(\mathtt{X}, \mathtt{gaussian}(2, 8)) :: \mathtt{temp}(\mathtt{D}, \mathtt{X})$, for example, states that the temperature for day `D` is Gaussian-distributed with mean 2 and standard deviation 8. The syntax allows one to specify multivariate distributions, i.e.,

$$((\mathtt{X}, \mathtt{Y}), \mathtt{gaussian}([1, 0], [[1, 0.5], [0.5, 1]]) :: \mathtt{f}(\mathtt{X}, \mathtt{Y}) \ .$$

In this paper, however, we restrict ourselves to the univariate case. From a user's perspective, continuous facts are queried like normal Prolog facts, and the value of the continuous variable is instantiated with a value drawn from the underlying distribution. Hybrid ProbLog adds the following predicates to the background knowledge of the theory to process values stemming from continuous facts:

- `below(X,`c`)` succeeds if `X` is a value from a continuous fact, c is a number constant, and $X < c$
- `above(X,`c`)` succeeds if `X` is a value from a continuous fact, c is a number constant, and $\mathtt{X} > c$
- `ininterval(X,`c_1`,`c_2`)` succeeds if `X` is a value from a continuous fact, c_1 and c_2 are number constants, and $\mathtt{X} \in [c_1, c_2]$

Unification with number constants is not supported for the values of continuous facts, i.e. the call $\mathtt{temp}(\mathtt{d}, 0)$ fails. But one can express the same using

$$\mathtt{temp}(\mathtt{d}, \mathtt{T}),\ \mathtt{ininterval}(\mathtt{T}, 0, 0) \ .$$

Similarly, standard Prolog comparison operators are not supported and one has to use the corresponding comparison predicate from the background knowledge:

$$\mathtt{temp}(\mathtt{d1}, \mathtt{T}),\ \mathtt{T} > 5$$

[4] We denote facts, values, and substitutions related to the continuous part of T by the superscript ^{c}.

has to be written as

$$\texttt{temp}(\texttt{d1},\texttt{T}),\ \texttt{above}(\texttt{T},5)\ .$$

Arithmetic expressions are not supported, i.e the query

$$\texttt{temp}(\texttt{d1},\texttt{T}),\ \texttt{Fahrenheit is}\ 9/5 * \texttt{X} + 32,\ \texttt{Fahrenheit} > 41$$

is illegal. There is no equivalent predicate for that in the background knowledge. Also, the comparison of two continuous facts is not supported, i.e. the query

$$\texttt{temp}(\texttt{d1},\texttt{T1}),\ \texttt{temp}(\texttt{d2},\texttt{T2}),\ \texttt{above}(\texttt{T1},\texttt{T2})$$

is illegal. The latter restriction, in particular, prevents two or more continuous variables getting "coupled", i.e. there is a dependency that requires one to always consider the values of both variables simultaneously. Furthermore, disallowing arithmetic expressions ensures that one can partition the underlying $\mathbb{R}^n$ space into simple intervals rather than into complex-shaped continuous sets. Despite being fairly restrictive, our framework allows for non-trivial programs.

Example 1 (Gaussian Mixture Model). The following theory encodes a Gaussian mixture model. The atom `mix(X)` can be used later on as if it were a simple continuous fact, which means the variable `X` can be processed using `above/2`, `below/2` and `ininterval/3`.

```
                    0.6::heads.        tails :- problog_not(heads).
(X, gaussian(0,1))::g(X).              mix(X) :- heads, g(X).
(X, gaussian(5,2))::h(X).              mix(X) :- tails, h(X).
```

The predicate `problog_not/1` is provided by the ProbLog inference engine. It allows one to negate an atom, similar to Prolog's \+, but can only be applied on *ground* probabilistic facts.

The following theory shall be used as running example throughout the paper to define the semantics of Hybrid Problog and to explain the inference algorithm.

Example 2 (Weather). This theory models weather during winter time. The background knowledge states that a person catches a cold when the temperature is below 0° Celsius or when it is colder than 5° Celsius while it rains.

```
                  0.8::rain.       catchcold :- rain, temp(T), below(T,5).
(T, gaussian(2,8))::temp(T).       catchcold :- temp(T), below(T,0).
```

The semantics of Hybrid ProbLog theory $T = F \cup F^c \cup \mathcal{BK}$ is given by probability distributions over subsets of the facts f_i (called *subprograms*), and over sample values for the numeric variables in the continuous facts f_i^c (called *continuous subprograms*). The subprograms $L \subseteq L_F$ are distributed as in ProbLog (cf. Equation (1)), and the continuous subprograms are distributed as described in Section 3.1. Combining both gives one the success probability of queries in a Hybrid ProbLog theory as described in Section 3.2.

3.1 Distribution over Continuous Subprograms

Let $\Theta^c = \{\theta^c_{j1}, \dots \theta^c_{ji'_j} | j = 1, \dots, m\}$ be a finite set of possible substitutions for the non-numeric variables in the continuous facts $(X_j, \phi_j) :: f^c_j$ where i'_j is the number of substitutions for fact j. Each substitution instance $f^c_j\theta^c_{jk}$ is associated with a random variable X_{jk} with probability distribution ϕ_j. The X_{jk} are assumed to be independent. Let $\boldsymbol{X}$ denote the $|\Theta^c|$-dimensional vector of the random variables, and $f(\boldsymbol{x})$ their joint density function. A sample value $\boldsymbol{x}$ for $\boldsymbol{X}$ defines the continuous subprogram $L_{\boldsymbol{x}} := \{f^c_j\theta^c_{jk}\{X_{jk} \leftarrow x_{jk}\} \mid j = 1, \dots, m; k = 1, \dots i'_j\}$ where $\{X_{jk} \leftarrow x_{jk}\}$ is the substitution of X_{jk} by x_{jk}.

Example 3 (Continuous Subprogram). Consider the following set of continuous facts where the second fact is *non-ground*. That is, one can obtain several ground instances where each instance has a continuous value drawn independently from the same distribution.

$$(\mathtt{X}, \mathtt{gaussian}(1, 2)) :: \mathtt{h}(\mathtt{X}). \qquad (\mathtt{X}, \mathtt{gaussian}(4, 3)) :: \mathtt{f}(\mathtt{X}, \mathtt{Y}).$$

When one applies the substitutions $\theta^c_{1,1} = \varnothing$, $\theta^c_{2,1} = \{\mathtt{Y} \leftarrow a\}$, $\theta^c_{2,2} = \{\mathtt{Y} \leftarrow b\}$ together with the point $x_{1,1} = 0.9$, $x_{2,1} = 2.3$, $x_{2,2} = 4.2$, one obtains the *continuous subprogram* $L_{\boldsymbol{x}} = \{\mathtt{h}(0.9), \mathtt{f}(2.3, \mathtt{a}), \mathtt{f}(4.2, \mathtt{b})\}$.

The joint distribution of $\boldsymbol{X}$ thus defines a distribution over continuous subprograms. Specifically, for a $|\Theta^c|$-dimensional interval $I = [a_{\theta^c_{11}}, b_{\theta^c_{11}}] \times \ldots \times [a_{\theta^c_{mi'_m}}, b_{\theta^c_{mi'_m}}]$ (which may also be open or half-open in every dimension), one obtains the probability of the set of continuous subprograms with continuous parameters in I:

$$P_P(\boldsymbol{X} \in I | F^c) := \int_{a_{\theta^c_{11}}}^{b_{\theta^c_{11}}} \cdots \int_{a_{\theta^c_{mi'_m}}}^{b_{\theta^c_{mi'_m}}} f(\boldsymbol{x})\, d\boldsymbol{x} \tag{3}$$

Example 4 (Joint Density). In Example 3, the joint density function is $f(\boldsymbol{x}) = f(x_{1,1}, x_{2,1}, x_{2,2}) = \varphi_{1,2}(x_{1,1}) \times \varphi_{4,3}(x_{1,2}) \times \varphi_{4,3}(x_{2,2})$ where $\varphi_{\mu;\sigma}$ is the density of a normal distribution $\mathcal{N}(\mu, \sigma)$.

3.2 Success Probabilities of Queries

The success probability $P_s(q)$ of a query q is the probability that q is provable in $L \cup L_{\boldsymbol{x}} \cup \mathcal{BK}$, where L is distributed according to (1), and $\boldsymbol{x}$ according to $f(\boldsymbol{x})$ respectively. The key to computing success probabilities is the consideration of admissible intervals, as introduced in the following definition.

Definition 1. *An interval $I \subseteq \mathbb{R}^{|\Theta^c|}$ is called* admissible *for a query q and a theory $T = F \cup F^c \cup \mathcal{BK}$ iff*

$$\forall \boldsymbol{x}, \boldsymbol{y} \in I, \forall L \subseteq L_F : \Big(L \cup L_{\boldsymbol{x}} \cup \mathcal{BK}\Big) \models q \Leftrightarrow \Big(L \cup L_{\boldsymbol{y}} \cup \mathcal{BK}\Big) \models q \tag{4}$$

If (4) *holds, we can also write* $L \cup L_I \cup \mathcal{BK} \models q$.

A partition $\mathcal{A} = I_1, I_2, \ldots, I_k$ *of* $\mathbb{R}^{|\Theta^c|}$ *is called* admissible *for a query* q *and a theory* T *iff all* I_i *are admissible intervals for* q *and* T.

In other words, an admissible interval I is "small enough" such that the values of the continuous variables, as long as they are in I, do not influence the provability of q. Within an admissible interval, the query either always fails or always succeeds for any sampled subset $L \subseteq L_F$ of probabilistic facts.

Example 5 (Admissible Intervals). Consider the Hybrid ProbLog theory consisting out of a single continuous fact:

$$(\texttt{X}, \texttt{gaussian}(\texttt{1}, \texttt{2})) :: \texttt{h}(\texttt{X})$$

For the query $\texttt{h}(\texttt{X}), \texttt{ininterval}(\texttt{X}, \texttt{5}, \texttt{10})$, the interval $[0, 10]$ is not admissible in this theory. The reason is, that for $x = 4 \in [0, 10]$ the query fails but for $x = 6 \in [0, 10]$ it succeeds. The intervals $[6, 9)$, $[5, 10]$, or $(-\infty, 5)$, for example, are all admissible. Note, that the inference engine allows one to evaluate conjunctive queries and that the predicate `ininterval/3` is automatically added to the background knowledge.

Definition 2 (Discretized Theory). *Let* $T = F \cup F^c \cup \mathcal{BK}$ *be a Hybrid ProbLog theory, then the* discretized theory T_D *is defined as*

$$\begin{aligned} & F \cup \{ f^c\{X \leftarrow f^c\} \mid (X, \phi) :: f^c \in F^c \} \\ & \quad \cup \mathcal{BK} \\ & \quad \cup \{ \texttt{below}(\texttt{X}, \texttt{C}), \texttt{above}(\texttt{X}, \texttt{C}), \texttt{ininterval}(\texttt{X}, \texttt{C1}, \texttt{C2}) \} \end{aligned}$$

where $f^c\{X \leftarrow f^c\}$ *is the atom resulting from substituting the variable* X *by the term* f^c.

The substitutions simplify the inference process. Whenever a continuous variable is used in a comparison predicate, the variable will be bound to the original continuous fact. Therefore, one can use a standard proof algorithm without keeping track of continuous variables. The discretized theory still contains probabilistic facts if F is not empty, thus it is a ProbLog theory. Definition 2 allows one to merge the infinite number of proofs, which every Hybrid ProbLog theory has, into a potentially finite number of proofs. This property is needed to compute the admissible intervals efficiently.

Example 6 (Proofs in the discretized theory). The discretized theory T_D for Example 2 is

```
0.8::rain.                 catchcold :- rain, temp(T), below(T, 5).
temp(temp(T)).             catchcold:- temp(T), below(T, 0).
below(X, C).               above(X, C).        ininterval(X, C1, C2).
```

The discretized theory contains two proofs for the query `catchcold`. For each proof, one can extract

– f_i the probabilistic facts used in the proof
– c_i the continuous facts used in the proof
– d_i the comparison operators used in the proof

The proofs of `catchcold` can be characterized by:

$$f_1 = \{\texttt{rain}\} \qquad c_1 = \{\texttt{temp(temp(T))}\} \qquad d_1 = \{\texttt{below(temp(T), 5)}\}$$
$$f_2 = \varnothing \qquad c_2 = \{\texttt{temp(temp(T))}\} \qquad d_2 = \{\texttt{below(temp(T), 0)}\}$$

It is possible, though not in Example 6, that the same continuous fact is used by several comparison operators within one proof, i.e. $f_i = \{\texttt{below(f(X), 10)}, \texttt{above(f(X), 0)}\}$. In such cases, one has to build the intersection of all intervals to determine the interval in which all comparison operators succeed, i.e. $f_i = \{\texttt{ininterval(f(X), 0, 10)}\}$. If that intersection is empty, the proof will fail in the original non-discretized theory. Building the intersections can also be interleaved with proving the goal.

The following theorem guarantees that an admissible partition exists for each query which has finitely many proofs in the discretized theory.

Theorem 1. *For every theory T, every query q that has only finitely many proofs in T_D, and all finite sets of possible substitutions for the probabilistic facts and the continuous facts Θ, Θ^c, there exists a finite partition of $\mathbb{R}^{|\Theta^c|}$ that is admissible for T and q.*

Proof. This follows from the fact that conditions defined by `below/2`, `above/2`, and `ininterval/3` are satisfied by intervals of sample values, and finite combinations of such conditions, which may appear in a proof, still define intervals. □

Algorithm 2 can be used to find *admissible partitions*. The algorithm has to be modified as described in Section 4 to respect interval endpoints. Given an admissible partition $\mathcal{A}$ one obtains the success probability of a query q as follows:

$$P_{s,\mathcal{A}}(q|T) := \sum_{L \subseteq L_F} \sum_{\substack{I \in \mathcal{A}: \\ L \cup L_I \cup \mathcal{BK} \models q}} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I | F^c) \tag{5}$$

The following theorem shows that the values of P_s are independent of the partition $\mathcal{A}$ and therefore we can write $P_s(q|T)$ instead of $P_{s,\mathcal{A}}(q|T)$.

Theorem 2. *Let $\mathcal{A}$ and $\mathcal{B}$ be admissible partitions for the query q and the theory T then $P_{s,\mathcal{A}}(q|T) = P_{s,\mathcal{B}}(q|T)$.*

Proof. Proven directly, by showing that for two admissible partitions one can construct a third partition that returns the same success probability. Using the definition for the success probability (5) we get:

$$P_{s,\mathcal{A}}(q|T) := \sum_{L \subseteq L_F} \sum_{\substack{I \in \mathcal{A}: \\ L \cup L_I \cup \mathcal{BK} \models q}} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I | F^c) \tag{6}$$

$$P_{s,\mathcal{B}}(q|T) := \sum_{L \subseteq L_F} \sum_{\substack{I \in \mathcal{B}: \\ L \cup L_I \cup \mathcal{BK} \models q}} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I | F^c) \tag{7}$$

Since $\mathcal{A}$ and $\mathcal{B}$ are both finite, one can construct a partition $\mathcal{C}$ such that it subsumes both $\mathcal{A}$ and $\mathcal{B}$, that is

$$\forall I \in \mathcal{A} : \exists I_1, \ldots, I_n \in \mathcal{C} : I = I_1 \cup \ldots \cup I_n \text{ and}$$
$$\forall I \in \mathcal{B} : \exists I'_1, \ldots, I'_{n'} \in \mathcal{C} : I = I'_1 \cup \ldots \cup I'_{n'}$$

Because $\mathcal{A}$ is admissible and by construction of $\mathcal{C}$, we can represent any summand in (6) as a sum over intervals in $\mathcal{C}$. That is, for each $L \subseteq L_F$ and each $I \in \mathcal{A}$ there exist $I_1, \ldots, I_n \in \mathcal{C}$ such that

$$P_P(L|F) \cdot P_P(\boldsymbol{X} \in I|F^c) = \sum_{i=1}^{n} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I_i|F^c) \ . \tag{8}$$

Because $\mathcal{A}$ is a partition and by construction of $\mathcal{C}$, the intervals needed to cover $I \in \mathcal{A}$ are disjoint from the intervals needed to cover $I' \in \mathcal{A}$ if $I \neq I'$. Therefore

$$\sum_{\substack{I \in \mathcal{A}: \\ L \cup L_I \cup \mathcal{BK} \models q}} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I|F^c) = \sum_{\substack{I \in \mathcal{C}: \\ L \cup L_I \cup \mathcal{BK} \models q}} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I|F^c) \tag{9}$$

for any subprogram $L \subseteq L_F$. From (9) and the definition of the *success probability* (5) follows

$$P_{s,\mathcal{A}}(q|T) = P_{s,\mathcal{C}}(q|T) \ .$$

Similarly, one can show that

$$P_{s,\mathcal{B}}(q|T) = P_{s,\mathcal{C}}(q|T) \ .$$

□

Theorem 1 and 2 guarantee that the semantics of Hybrid ProbLog, i.e. the fragment that restricts the usage of continuous values, is well-defined. The imposed restrictions provide a balance between expressivity and tractability. They allow one to discretize the space $\mathbb{R}^n$ of possible assignments to the continuous facts in multidimensional intervals such that the actual values within one interval do not matter. In turn, this makes efficient inference algorithms possible. Comparing two continuous values against each other would couple them. This would require a more complicated discretization of $\mathbb{R}^n$ in the form of polyhedra which are harder to represent and to integrate over. Allowing arbitrary functions to be applied on continuous values, eventually, leads to a fragmentation of the space in arbitrary sets. This makes exact inference virtually intractable.

4 Exact Inference

In this section we present an exact inference algorithm for Hybrid ProbLog. Our approach generalizes De Raedt *et al.*'s BDD algorithm [8] and generates a BDD [5] that is evaluated by a slight modification of the original algorithm. The pseudocode is shown in Algorithm 1, 2 and 3. In the remainder of this section, we explain the inference steps on Example 2 and calculate the success probability of the query `catchcold`.

Algorithm 1 The inference algorithm collects all possible proofs and partitions the $\mathbb{R}^n$ space according to the constraints imposed by each proof. The intermediate variables f'_u and c'_u are superfluous and have been added to simplify the explanations in this section.

1: **function** SUCCESSPROB(query q, theory T)
2: $\{(f_i, c_i, d_i)\}_{1 \le i \le m} \leftarrow$ FINDALLPROOFS(T, q) ▷ Backtracking in Prolog
3: **for** $c\theta \in \cup_{1 \le i \le m} c_i$ **do** ▷ Iterate over used ground continuous facts
4: $\mathbf{A}_{c\theta} \leftarrow$CREATEPARTITION$(\{d_1, \ldots, d_m\})$
5: $\{b_{c\theta,I}\}_{I \in A_{c\theta}} \leftarrow$CREATEAUXBODIES$(\mathbf{A}_{c\theta})$
6: **end for**
7: $u \leftarrow 0$ ▷ # disjoint proofs
8: **for** $i = 1, 2, \ldots, m$ **do** ▷ Go over all proofs
9: $(\hat{c}_1\hat{\theta}_1, \ldots, \hat{c}_t\hat{\theta}_t) \leftarrow c_i$ ▷ All cont. facts of proof i (this simplifies notation)
10: $(\hat{d}_1, \ldots, \hat{d}_t) \leftarrow d_i$ ▷ Intervals in proof i (this simplifies notation)
11: **for** $(I_1, \ldots, I_t) \in \mathbf{A}_{\hat{c}_1\hat{\theta}_1} \times \cdots \times \mathbf{A}_{\hat{c}_t\hat{\theta}_t}$ **do** ▷ Go over all possible intervals
12: **if** $(d_1 \cap I_1 \neq \varnothing) \ \wedge \ \ldots \ \wedge (d_t \cap I_t \neq \varnothing)$ **then**
13: $u \leftarrow u + 1$ ▷ Add one more disjoint proof
14: $f'_u \leftarrow f_i$ ▷ Probabilistic facts stay the same
15: $c'_u \leftarrow c_i$ ▷ Continuous facts stay the same
16: $d'_u \leftarrow \{\mathrm{dom}(\hat{c}_1\hat{\theta}_1) = I_1, \ldots, \mathrm{dom}(\hat{c}_t\hat{\theta}_t) = I_t\}$ ▷ Domains are adapted
17: $f''_u \leftarrow f_i \cup \{b_{c\theta,I} | c\theta \in c'_u, I \in d'_u\}$ ▷ Add aux. bodies to the facts
18: **end if**
19: **end for**
20: **end for**
21: $BDD \leftarrow$GENERATEBDD$(\bigvee_{1 \le i \le u} \bigwedge_{f \in f''_i} f)$ ▷ cf. [9]
22: **return** PROB(root(BDD)) ▷ cf. [8]
23: **end function**

1. All proofs for `catchcold` are collected by SLD resolution (Line 2 in Algorithm 1).

$$f_1 = \{\texttt{rain}\} \qquad c_1 = \{\texttt{temp(T)}\} \qquad d_1 = \{\texttt{T} \in (-\infty, 5)\}$$
$$f_2 = \varnothing \qquad c_2 = \{\texttt{temp(T)}\} \qquad d_2 = \{\texttt{T} \in (-\infty, 0)\}$$

   Each proof is described by a set of probabilistic facts f_i, a set of continuous facts c_i, and an interval for each continuous variable in c_i. When a continuous fact is used within a proof, it is added to c_i and the corresponding variable X is added to d_i with $X \in (-\infty, \infty)$.

   When later on `above(X,`c_1`)` is used in the same proof, the interval I stored in d_i is replaced by $I \cap (c_1, \infty)$, similarly for `below(X,`c_2`)` it is replaced by $I \cap (-\infty, c_2)$, and for `ininterval(X,` $\texttt{c}_1$`,` $\texttt{c}_2$`)` it is replaced by $I \cap [c_1, c_2]$,

2. We partition $\mathbb{R}^1$ because one continuous fact is used. The loop in Line 3 of Algorithm 1 iterates over $(\cup_{1 \le i \le m} c_i) = \{\texttt{temp(T)}\}$. When calling the function CREATEPARTITION$(\{d_1, d_2\})$ (cf. Algorithm 2) we obtain the admissible partition $\{(-\infty, 0),\ [0, 5),\ [5, \infty)\}$ which is used to disjoin the proofs with

Algorithm 2 This function returns a partition of $\mathbb{R}$ by creating intervals touching the given intervals. Partitions of $\mathbb{R}^n$ can be obtain by building the cartesian product over the partitions for each dimension. This is possible due to the restrictions imposed in Section 3.

```
1: function CreatePartition(Set of intervals D = {d_1, ... d_m})
2:     C ← ⋃_{i=1}^{m} {low_i, high_i}        ▷ low_i and high_i are interval endpoints of d_i
3:     C ← C ∪ {−∞, ∞}                        ▷ add upper and lower limit of ℝ
4:     (c'_1, ..., c'_k) ← SortAndIgnoreDuplicates(c)
5:     Result ← { (−∞, c'_2], (c'_{k−1}, ∞) }        ▷ c'_1 = −∞ and c'_k = ∞
6:     for i = 2, ..., k − 1 do
7:         Result ← Result ∪ { (c'_{i−1}, c'_i] }
8:     end for
9:     return Result
10: end function
```

respect to the continuous facts (Line 8-20 in Algorithm 1):

$$
\begin{array}{lll}
f'_1 = \{\texttt{rain}\} & c'_1 = \{\texttt{temp(T)}\} & d'_1 = \{\texttt{T} \in (-\infty, 0)\} \\
f'_2 = \{\texttt{rain}\} & c'_2 = \{\texttt{temp(T)}\} & d'_2 = \{\texttt{T} \in [0, 5)\} \\
f'_3 = \varnothing & c'_3 = \{\texttt{temp(T)}\} & d'_3 = \{\texttt{T} \in (-\infty, 0)\}
\end{array}
$$

3. We create one auxiliary fact per continuous fact and interval. They are dependent, i.e. $\texttt{temp(T)}_{[-\infty,0)}$ and $\texttt{temp(T)}_{[0,5)}$ cannot be true at the same time. We have to make the dependencies explicit by adding, conceptually, the following clauses to the theory and replacing calls to continuous facts and background predicates by the bodies $b_{c\theta,I}$:

$$
\begin{aligned}
\texttt{call\_temp(T)}_{(-\infty,0)} &\texttt{ :- temp(T)}_{(-\infty,0)} \\
\texttt{call\_temp(T)}_{[0,5)} &\texttt{ :- } \neg\texttt{temp(T)}_{(-\infty,0)}, \texttt{temp(T)}_{[0,5)} \\
\texttt{call\_temp(T)}_{[5,\infty)} &\texttt{ :- } \neg\texttt{temp(T)}_{(-\infty,0)}, \neg\texttt{temp(T)}_{[0,5)}, \texttt{temp(T)}_{[5,\infty)}
\end{aligned}
$$

Only one of the clauses can be true at the same time. The bodies encode a linear chain of decisions. The probability attached to an auxiliary fact $\texttt{temp(T)}_{[l,h)}$ is the conditional probability that the sampled value of T is in the interval $[l, h)$ given it is not in $(-\infty, l)$

$$
P\left(\texttt{temp(T)}_{[l,h]}\right) := \left[\int_l^h \mathcal{N}(2, 8, x)\, dx\right] \cdot \left[1 - \int_{-\infty}^{l} \mathcal{N}(2, 8, x)\, dx\right]^{-1} \tag{10}
$$

where $\mathcal{N}$ is the density of the Gaussian specified for temp/1 in the program. This encodes a switch (cf. [11]) such that the success probability of $\texttt{call\_temp(T)}_{[l,h]}$ is exactly $\int_l^h \mathcal{N}(2, 8, x)\, dx$. To evaluate the cumulative density function, we use the function Phi as described in [10]. If we want to use any other distribution, we have to only replace the evaluation function of the

Algorithm 3 To evaluate the BDD we run a modified version of De Raedt *et al.*'s algorithm that takes the conditional probabilities for each continuous node into account. For Gaussian-distributed continuous facts we use the function PHI [10] to evaluate $\int_{l_i}^{h_i} \mathcal{N}(x, \mu_i, \sigma_i)\, dx$ which performs a Taylor approximation of the cumulative density function (CDF). If the program uses distributions other than Gaussians, the user has to provide the corresponding CDF.

```
1: function PROB(node n)
2:     if n is the 1-terminal then return 1
3:     if n is the 0-terminal then return 0
4:     Let h and l be the high and low children of n
5:     p_h ← PROB(h)
6:     p_l ← PROB(l)
7:     if n is a continuous node with attached interval [a, b] and density φ_n then
8:         p ← [∫_{l_n}^{h_n} φ_n(x) dx] · [1 − ∫_{−∞}^{l_n} φ_n(x) dx]^{−1}
9:     else
10:        p ← p_n        ▷ the probability attached to the fact in the ProbLog program
11:    end if
12:    return p · p_h + (1 − p) · p_l
13: end function
```

density, as all the rest of the algorithm does not depend on the particular distribution. Adding the bodies of the auxiliary clauses to f_i' yields the final set of proofs (Line 17 in Algorithm 1):

$$f_1'' = \{\texttt{rain}, \texttt{temp(T)}_{(-\infty,0)}\}$$
$$f_2'' = \{\texttt{rain}, \neg\texttt{temp(T)}_{(-\infty,0)}, \texttt{temp(T)}_{[0,5)}\}$$
$$f_3'' = \{\texttt{temp(T)}_{(-\infty,0)}\}$$

The proofs are now disjoint with respect to the continuous facts. That is, either the intervals for continuous facts are disjoint or identical. With respect to the probabilistic facts, they are not disjoint and summing up the probabilities of all proofs would yield a wrong result. One would count the probability mass of the overlapping parts multiple times [8].

4. To account for that, we translate the proofs into a Boolean expression in disjunctive normal form (cf. Line 21 in Algorithm 1) and represent it as BDD (cf. Figure 1). This step is similar to ProbLog's inference mechanism

$$\Big(\texttt{rain} \wedge \texttt{temp(T)}_{(-\infty,0)}\Big)$$
$$\vee\Big(\texttt{rain} \wedge \neg\texttt{temp(T)}_{(-\infty,0)} \wedge \texttt{temp(T)}_{[0,5)}\Big)$$
$$\vee\Big(\texttt{temp(T)}_{(-\infty,0)}\Big)$$

5. We evaluate the BDD with Algorithm 3 and get the success probability of `catchcold`. This is a slight modification of De Raedt *et al.*'s algorithm (cf. [8, 9] for the details) that takes into account the continuous nodes.

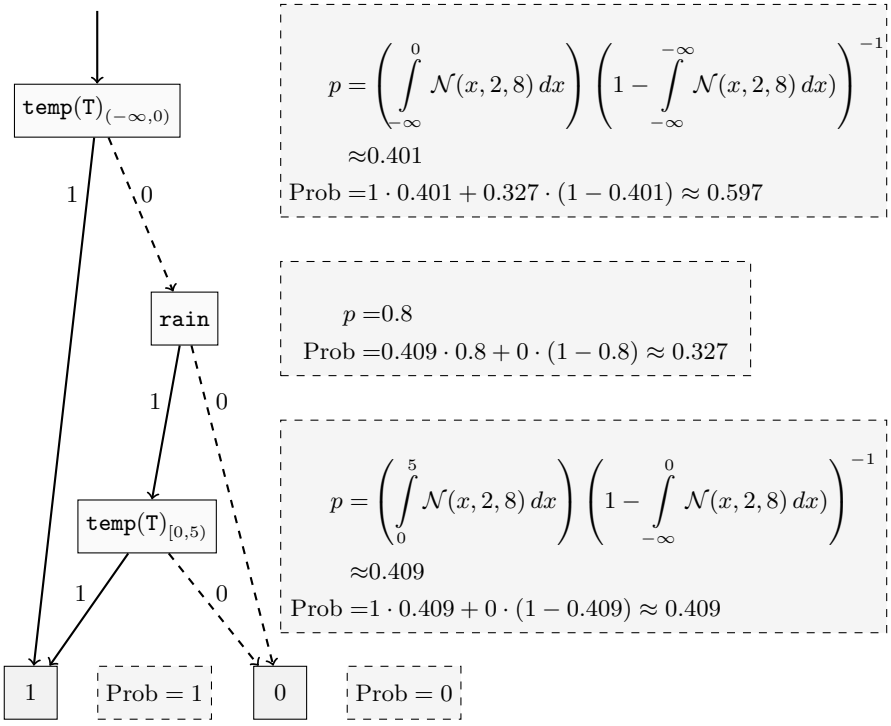

Fig. 1. This BDD [5] encodes all proofs of `catchcold` in the theory from Example 2. The dashed boxes show the intermediate results while traversing the BDD with Algorithm 3. The success probability of the query is returned at the root and is 0.597.

The function CreatePartition (cf. Algorithm 2) does not necessarily return an *admissible partition* as it ignores the interval endpoints by creating right-open intervals. For instance, if one obtains two proofs which impose as constraint the intervals $[1, 2]$ and $[2, 3)$, the minimal admissible partition is

$$\{(-\infty, 1), [1, 2), [2, 2], (2, 3), [3, \infty)\} \ .$$

The function CreatePartition, however, returns the inadmissible partition

$$\{(-\infty, 1), [1, 2), [2, 3), [3, \infty)\} \ .$$

With respect to the interval $[2, 3)$, the first proof succeeds for $x = 2$ but fails for any other value in $[2, 3)$ – which is not allowed for admissible intervals (cf. Definition 1). Though, when calculating the *success probability*, one can ignore interval endpoints. Since the calculation involves integrals of the form $\int_{l_i}^{h_i} \phi(x)\, dx$, it is irrelevant whether intervals are open or closed (cf. Equation (10)). Also, an integral over a single point interval has value 0.

However, when one wants to know whether there is a proof with specific values for some or all continuous facts, one has to be precise about the interval

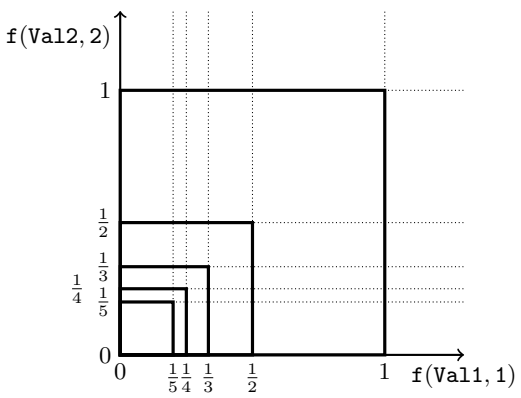

Fig. 2. The query $\mathtt{s}(5,2)$ succeeds, if both values $\mathtt{f}(\mathtt{Val1},1)$ and $\mathtt{f}(\mathtt{Val2},2)$ lie in one of the intervals $[0,\frac{1}{1}]$, $[0,\frac{1}{2}]$, ..., $[0,\frac{1}{5}]$. These areas correspond to the thick-lined squares starting at $(0,0)$. Since they overlap, one has to partition the space $\mathbb{R}^2$ in order to disjoin the proofs. Due to the restrictions of Hybrid ProbLog program, i.e., continuous variables cannot be compared against each other, one can obtain an admissible partition for each dimension independently. Algorithm 2 returns the partitions shown by the dotted lines. The horizontal lines partition the space of $\mathtt{f}(\mathtt{Val2},2)$ and the vertical the space of $\mathtt{f}(\mathtt{Val1},1)$.

endpoints and use a modified algorithm. Admissibility can be ensured, for instance, by creating for each pair of constants the open interval (l_i, h_i) and the single point interval $[l_i, h_i]$. For the former example this is

$$\{(-\infty,1),[1,1],(1,2),[2,2],(2,3),[3,3],(3,\infty)\} \ .$$

5 Experiments

We implemented Hybrid ProbLog in YAP 6.0.7 and set up experiments to answer the question

> How does the inference algorithm (cf. Algorithm 1) scale in the size of the partitions and in the number of ground continuous facts?

In domains where exact inference is feasible, the number of continuous facts and comparison operations is typically small compared to the rest of the theory. Our algorithm is an intermediate step between SLD resolution and BDD generation. Therefore, it is useful to know how much the disjoining operations cost compared to the other inference steps. We tested our algorithm on the following theory:

```
(Val, gaussian(0,1)) :: f(Val, ID).
    s(Consts, Facts) :- between(1, Facts, ID), between(1, Consts, Top),
                        High is Top/Consts,
                        f(Val, ID), ininterval(Val, 0, High).
```

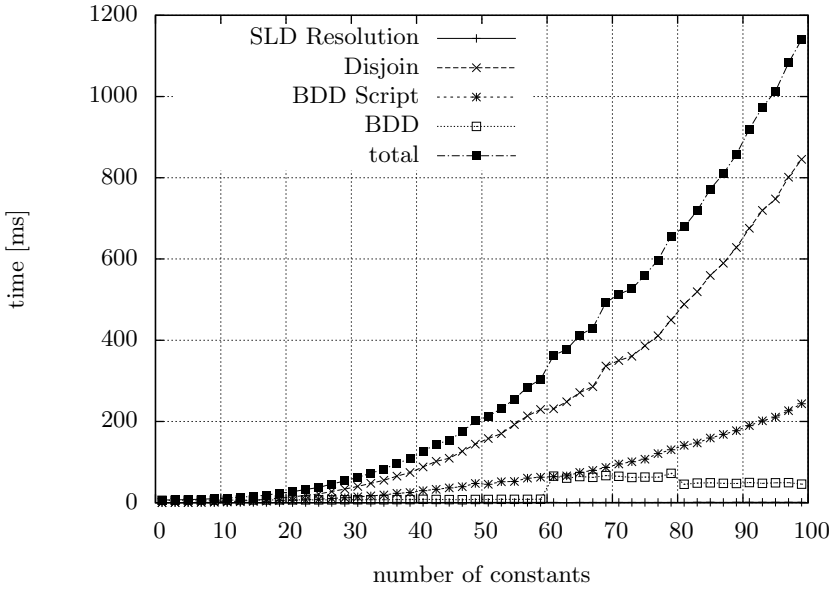

Fig. 3. Runtimes for calculating the success probability of $\mathtt{s}(\mathtt{Consts}, \mathtt{Facts})$ for varying the number of constants $s(1,1), \cdots s(100,1)$. As the graphs show, most of the time is spent on disjoining the proofs, that is partitioning the domains. The BDD time stays more or less constant, this is due to the fact that the resulting Boolean expression, i.e. $\neg \mathtt{s}(\mathtt{Val},1)_{(-\infty,0)} \wedge \left(\mathtt{s}(\mathtt{Val},1)_{[0,\frac{1}{n})} \vee \mathtt{s}(\mathtt{Val},1)_{[\frac{1}{n},\frac{1}{n-1})} \vee \ldots \vee \mathtt{s}(\mathtt{Val},1)_{[\frac{1}{2},1)}\right)$, is rather simple. This can be detected and exploited by the BDD package.

The query $\mathtt{s}(\mathtt{Consts}, \mathtt{Facts})$ has $\mathtt{Consts} \times \mathtt{Facts}$ many proofs where both arguments have to be positive integers. The first argument determines the number of partitions needed to disjoin the proofs with respect to the continuous facts and the second argument determines how many continuous facts are used. The query $\mathtt{s}(5,2)$, for instance, uses two continuous facts, $\mathtt{f}(\mathtt{Val1},1)$ and $\mathtt{f}(\mathtt{Val2},1)$, and compares them to the intervals $[0,1],[0,\frac{1}{2}],\cdots[0,\frac{1}{5}]$. Figure 2 shows the resulting partitioning when proving the query $\mathtt{s}(5,2)$. In general, one obtains $(\mathtt{Consts}+1)^{\mathtt{Facts}}$ many partitions of the space $\mathbb{R}^{\mathtt{Facts}}$.

The success probability of $\mathtt{s}(\mathtt{Consts}, \mathtt{Facts})$ is independent of $\mathtt{Consts}$. That is, for fixed $\mathtt{Facts}$ and any $c_1, c_2 \in \mathbb{N}$

$$P_s(\mathtt{s}(c_1, \mathtt{Facts})|T) = P_s(\mathtt{s}(c_2, \mathtt{Facts})|T)$$

We ran two series of queries[5]. First, we used one continuous fact and varied the number of constants from 1 to 100. As the graph in Figure 3 shows, the disjoin

[5] The experiments were performed on an Intel Core 2 Quad machine with 2.83GHz and 8GB of memory. We used the CUDD package for BDD operations and set the reordering heuristics to CUDD_REORDER_GROUP_SIFT. Each query has been evaluated 20 times and the runtimes were averaged.

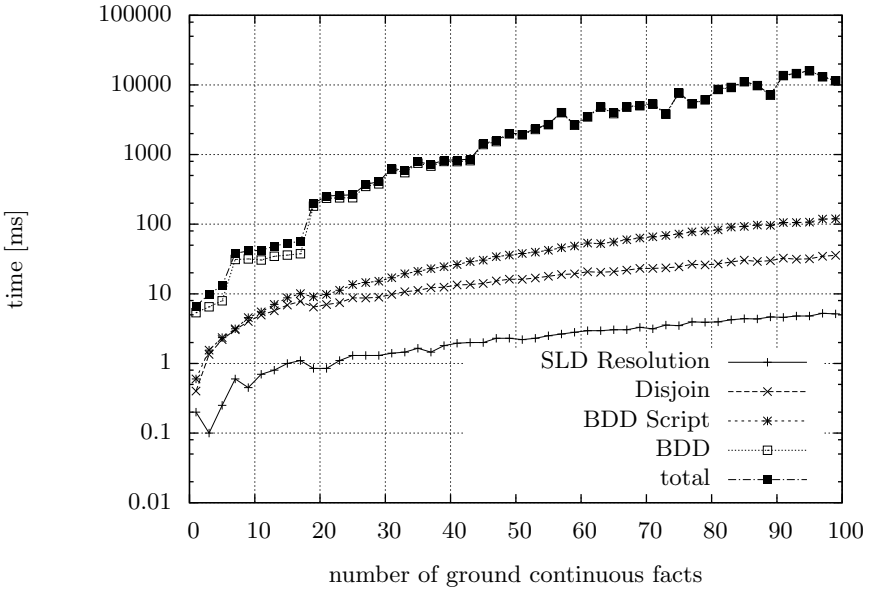

Fig. 4. Runtimes for calculating the success probability of $\mathtt{s}(\mathtt{Consts}, \mathtt{Facts})$ for varying the number of dimensions $s(5,1), \ldots, s(5,100)$. In this setting, most of the time is spent in the BDD package, that is building the BDD based on the script and traversing it. The runtime for our disjoin operation grows only linearly. This is due to the fact that the partitions of $\mathbb{R}^n$ can be factorized into each dimensions because we do not allow comparisons between two continuous variables.

operation – that is finding all partitions, generating the auxiliary bodies and rewriting the proofs – runs in $O(\mathtt{Consts}^2)$ when everything else stays constant. In this case, due to compression and pruning operations during the BDD script generation [9] (the input for the BDD package), building and evaluating the BDD runs in quasi-linear time. In the second run, we varied the number of continuous facts by evaluating the queries $\mathtt{s}(5,1), \cdots, \mathtt{s}(5,100)$. As Figure 4 shows, our algorithm (depicted by the Disjoin graph) runs in linear time. The runtime for the BDD operations grows exponentially due to the reordering heuristics used by the BDD package.

6 Conclusions and Future Work

We extended ProbLog with continuous distributions and introduced an exact inference algorithm. The expressivity has been restricted to make inference tractable. Possible directions for future work include comparisons between two continuous facts and applying functions on continuous values. We are working on upgrading existing ProbLog parameter learning methods to continuous facts. ProbLog is available for download at http://dtai.cs.kuleuven.be/problog.

Acknowledgments

Bernd Gutmann is supported by the Research Foundation-Flanders (FWO-Vlaanderen). This work is supported by the GOA project 2008/08 Probabilistic Logic Learning and by the European Community's Seventh Framework Programme under grant agreement First-MM-248258.

References

1. Vennekens, J., Denecker, M., Bruynooghe, M.: Representing causal information about a probabilistic process. In Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A., eds.: Logics in Artificial Intelligence, 10th European Conference, (JELIA 2006), Liverpool, UK. Volume 4160 of Lecture Notes in Computer Science., Springer (2006) 452–464
2. Sato, T.: A statistical learning method for logic programs with distribution semantics. In Sterling, L., ed.: Proceedings of the Twelfth International Conference on Logic Programming (ICLP 1995), MIT Press (1995) 715–729
3. Milch, B., Marthi, B., Russell, S.J., Sontag, D., Ong, D.L., Kolobov, A.: Blog: Probabilistic models with unknown objects. In Kaelbling, L.P., Saffiotti, A., eds.: IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005, Professional Book Center (2005) 1352–1359
4. Goodman, N., Mansinghka, V.K., Roy, D.M., Bonawitz, K., Tenenbaum, J.B.: Church: a language for generative models. In McAllester, D.A., Myllymäki, P., eds.: UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, July 9-12, 2008, Helsinki, Finland, AUAI Press (2008) 220–229
5. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers **35**(8) (1986) 677–691
6. Wang, J., Domingos, P.: Hybrid markov logic networks. In Fox, D., Gomes, C.P., eds.: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008, AAAI Press (2008) 1106–1111
7. Murphy, K.: Inference and learning in hybrid bayesian networks. Technical Report UCB/CSD-98-990, University of Berkeley (1998)
8. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In Veloso, M.M., ed.: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India. (2007) 2462–2467
9. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., De Raedt, L.: On the efficient execution of probLog programs. In Garcia de la Banda, M., Pontelli, E., eds.: Logic Programming. Volume 5366 of Lecture Notes in Computer Science. Springer Berlin/Heidelberg (2008) 175–189
10. Marsaglia, G.: Evaluating the normal distribution. Journal of Statistical Software **11**(5) (7 2004) 1–11
11. De Raedt, L., Demoen, B., Fierens, D., Gutmann, B., Janssens, G., Kimmig, A., Landwehr, N., Mantadelis, T., Meert, W., Rocha, R., Santos Costa, V., Thon, I., Vennekens, J.: Towards digesting the alphabet-soup of statistical relational learning. In Roy, D., Winn, J., McAllester, D., Mansinghka, V., Tenenbaum, J., eds.: Proceedings of the 1st Workshop on Probabilistic Programming: Universal Languages, Systems and Applications, Whistler, Canada (December 2008)