Radboud Repository

Radboud University Nijmegen

PDF hosted at the Radboud Repository of the Radboud University Nijmegen

The following full text is a preprint version which may differ from the publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/117051

Please be advised that this information was generated on 2019-02-19 and may be subject to change.

Inference for a New Probabilistic Constraint Logic*

Steffen Michels, Arjen Hommersom, Peter J.F. Lucas, Marina Velikova and **Pieter Koopman**
Radboud University Nijmegen
The Netherlands
s.michels@science.ru.nl

Abstract

Probabilistic logics combine the expressive power of logic with the ability to reason with uncertainty. Several probabilistic logic languages have been proposed in the past, each of them with their own features. In this paper, we propose a new *probabilistic constraint logic programming* language, which combines *constraint logic programming* with probabilistic reasoning. The language supports modeling of discrete as well as continuous probability distributions by expressing constraints on random variables. We introduce the declarative semantics of this language, present an exact inference algorithm to derive bounds on the joint probability distributions consistent with the specified constraints, and give experimental results. The results obtained are encouraging, indicating that inference in our language is feasible for solving challenging problems.

1 Introduction

Probabilistic logics combine the power of logic to represent knowledge with the ability to deal with uncertainty. In earlier research (e.g. [Bacchus, 1990]), this concerned logics with formulas such as $P^{>}_{0.5}\varphi$, meaning that the 'agent believes φ with probability strictly greater than 0.5'. While there has been significant work in this area, these types of logics are typically both computationally and conceptually complex. In recent years, powerful probabilistic logics have been developed that allow for efficient inference and learning (cf. [Getoor and Taskar, 2007] for an overview). Many of these languages are based on Sato's distribution semantics [Sato, 1995], which extends a joint probability distribution over the facts of a logic program to a distribution over the set of possible least models of the entire program. Examples of such languages are ProbLog [Raedt *et al.*, 2007], PRISM [Sato and Kameya, 1997], ICL [Poole, 2008], and CP-logic [Vennekens *et al.*, 2009].

Significant research has also been done to extend logic programming (LP) with constraints, which lead to *constraint logic programming* (CLP) [Jaffar and Lassez, 1987]. The general approach supports various constraint theories. For example, in CLP(FD) [Codognet and Diaz, 1996] constraints involving variables with a finite domain can be handled. Another example is CLP($\mathcal{R}$) [Jaffar *et al.*, 1992], which introduces constraints on real numbers inside LP.

In this paper, we combine probabilistic with constraint logic programming, called *probabilistic constraint logic programming* (PCLP), by associating a set of constraints with a probability distribution. The idea of modeling probabilistic information using constraints was previously explored in CLP($\mathcal{BN}$) [Costa and Cussens, 2003] and an identically named language [Reizler, 1998], which both use the CLP mechanism to model a joint probability distribution on the models of a logic program. The key difference is that instead of modeling a *single joint distribution*, we use PCLP to define *constraints on a distribution*. Thus, PCLP supports *imprecise probabilities*, meaning that some of the probabilistic information is unknown, e.g., as in some of the earlier probabilistic logics, it is possible to specify that the probability of a fact being true is larger than 0.5.

This way of defining distributions allows PCLP ito deal with continuous distributions in a powerful way. Although Hybrid ProbLog [Gutmann *et al.*, 2010] extends ProbLog with continuously distributed facts, the logical language involving these distributions is heavily restricted, e.g., it is not possible to compare two random variables. Another approach is the work on distributional clauses, which extends the expressiveness of ProbLog further [Gutmann *et al.*, 2011]. Although this language supports arbitrary parameterized distributions, inference is only feasible by sampling. In comparison, continuous distributions can be modeled in PCLP by providing probabilistic information about constraints in CLP($\mathcal{R}$), consistent with the actual distribution. The lower and upper bound of marginal probabilities can be derived using exact inference. The bounds can serve as approximation with desired precision, which depends on the amount of probabilistic information encoded by the constraints.

This paper is organized as follows. In the following section, PCLP is motivated by means of an example. In Sec-

*This publication was supported by the Dutch national program COMMIT. The research work was carried out as part of the Metis project under the responsibility of the Embedded Systems Institute with Thales Nederland B.V. as the carrying industrial partner. We omitted all proofs in this paper due to lack of space. An extended version including the proofs will be made available.

tion 3, we will formally introduce the syntax and semantics of PCLP. Section 4 introduces an inference procedure for PCLP, which we apply in Section 5 to the example. Finally, in Section 6, the achievements are summarized and we point out directions for future research.

2 Motivating Example

To illustrate the expressive power of the new language, we present an example on the likelihood consumers will buy a certain kind of fruit, based on [Binder *et al.*, 1997]. Since we have a first-order formalism, this generalizes easily to an arbitrary number of kinds (in the example: apples and bananas).

Yield of fruit is clearly relevant for the price. We model the yield of fruit with normally distributed random variables $X \sim \mathcal{N}(\mu, \sigma^2)$:

$$\text{Yield}(\mathit{apple}) \sim \mathcal{N}(12000.0, 1000.0)$$
$$\text{Yield}(\mathit{banana}) \sim \mathcal{N}(10000.0, 1500.0)$$

The price is also influenced by government support:

$$\text{Support}(\mathit{apple}) \sim \{0.3\colon \mathit{yes}, 0.7\colon \mathit{no}\}$$
$$\text{Support}(\mathit{banana}) \sim \{0.5\colon \mathit{yes}, 0.5\colon \mathit{no}\}$$

The basic price linearly depends on the yield:

$$\mathit{basic\_price}(\mathit{apple}, 250 - 0.007 \cdot \text{Yield}(\mathit{apple}))$$
$$\mathit{basic\_price}(\mathit{banana}, 200 - 0.006 \cdot \text{Yield}(\mathit{banana}))$$

In case the price is supported it is raised by a fixed amount:

$$\mathit{price}(\mathit{Fruit}, \mathit{BPrice} + 50) \leftarrow$$
$$\mathit{basic\_price}(\mathit{Fruit}, \mathit{BPrice}),\ \text{Support}(\mathit{Fruit}) = \mathit{yes}$$
$$\mathit{price}(\mathit{Fruit}, \mathit{BPrice}) \leftarrow$$
$$\mathit{basic\_price}(\mathit{Fruit}, \mathit{BPrice}),\ \text{Support}(\mathit{Fruit}) = \mathit{no}$$

At which maximum price customers still buy a certain fruit is modeled by a gamma distribution:

$$\text{Max\_price}(\mathit{apple}) \sim \Gamma(10.0, 18.0)$$
$$\text{Max\_price}(\mathit{banana}) \sim \Gamma(12.0, 10.0)$$

Thus, a customer is willing to buy in case the price is equal to or is less than the maximum price, which can be expressed by the following first-order rule:

$$\mathit{buy}(\mathit{Fruit}) \leftarrow \mathit{price}(\mathit{Fruit}, P),\ P \leq \text{Max\_price}(\mathit{Fruit})$$

Based on these constraints, the following probability approximations can be determined:

$$P(\mathit{buy}(\mathit{apple})) \approx 0.464 \pm 0.031$$
$$P(\mathit{buy}(\mathit{banana})) \approx 0.162 \pm 0.031$$
$$P(\mathit{buy}(\mathit{apple}) \vee \mathit{buy}(\mathit{banana})) \approx 0.552 \pm 0.054$$

The actual probabilities are guaranteed to be within the computed maximum error determined by the constraints and used approximation scheme.

Besides using the language for approximations, it can also express imprecise probabilities. For instance, to express that the probability that the price of apples is supported is in the interval $[0.3, 1]$, we can write:

$$\text{Support}(\mathit{apple}) \sim \{0.3\colon \mathit{yes}, 0.7\colon \mathit{yes} \vee \mathit{no}\}$$

The bounds on the probability that the consumer buys apples is now $0.222 \leq P(\mathit{buy}(\mathit{apple})) \leq 0.493$. These bounds represent both the underspecified distribution and the error in approximating the continuous distributions.

3 The PCLP Language

The *probabilistic constraint logic program* (PCLP) language $\mathcal{L}$ consists of a constraint language $\mathcal{C}$, the language of rules $\mathcal{R}$, and definitions of random variables $\mathcal{V}$. Below, we assume the language to be fixed. Then, a PCLP $\mathcal{T}$ (a "theory") consists of random variable definitions $\mathbf{V}_\mathcal{T}$, and a logical theory $\mathbf{L}_\mathcal{T} = \mathbf{C}_\mathcal{T} \cup \mathbf{R}_\mathcal{T}$, where $\mathbf{C}_\mathcal{T}$ is a constraint theory, related to the used constraints domains as discussed hereafter, and $\mathbf{R}_\mathcal{T}$ is a set of acyclic rules.

3.1 Syntax

Constraint Domains

In this paper, we use a language of constraints that is closed under conjunction and negation. We explore two constraint domains which are commonly used in constraint logic programming — discrete values ($\mathbb{DV}$) and the real numbers ($\mathbb{R}$) — to model constraints on discrete and continuous random variables, respectively.

In the constraint domain $\mathbb{DV}$ random variables take discrete values. The constraint language consists of equality ($=$) and inequality ($\neq$) as basic building blocks. In the example we used constants such as yes in the definitions as abbreviation for the constraint that a random variable equals that constant. The constraint domain is very similar to the finite constraint domain CLP(FD) [Hentenryck, 1989], with the difference that we do not require the domain to be defined explicitly. The domain of all variables is implicitly given by all atoms occurring in the program, but one can exclude values by not assigning any probability mass to them. Using $\mathbb{DV}$, one can represent, for instance, Bayesian networks [Pearl, 1988], but also first-order formalisms such as CP-logic [Vennekens *et al.*, 2009].

The $\mathbb{R}$ domain is basically the same as in CLP($\mathcal{R}$) [Jaffar *et al.*, 1992]. Variables represent real numbers and constraints consist of linear equalities and inequalities using predicates such as $\{=, \neq, <, >, \leq, \geq\}$ and functions $\{+, -, *\}$. This theory can be used to approximate arbitrary continuous distributions for which the *cumulative distribution function* is known by associating probabilities to intervals of the domain of the distribution, defining a set of distributions including the actually intended one. To do that one can divide the probability space of a single variable in n intervals $(l_1, u_1), \cdots, (l_n, u_n)$ and constrain its distribution as $\text{X} \sim \{P(l_1 < \text{X} < u_1)\colon l_1 < \text{X} < u_1, \cdots, P(l_n < \text{X} < u_n)\colon l_n < \text{X} < u_n\}$. The number of intervals determine the precision, i.e. the gap between the probability bounds one can compute.

Rule Language

Rules are (implicitly universally quantified) expressions of the form: $h \leftarrow l_1, \ldots, l_n$ where h is called the *head* and the conjunction $l_1, \ldots, l_n$ is called the *body* of the rule. The head h is an *atom*, i.e., an expression of the form $p(t_1, \ldots, t_m)$ with p a *predicate*, and $l_1, \ldots, l_n$ are either atoms or constraints (elements of $\mathcal{C}$). Constants are denoted by lower-case letters (a, b, ...), while variables start with upper-case letters (e.g. X, Y, ...). As mentioned above, $\mathbf{R}_\mathcal{T}$ is defined as the set of rules of the PCLP theory $\mathcal{T}$.

Random Variable Definitions

Each random variable definition is of the form: $V(t_1, \ldots, t_n) \sim \{p_1 \colon c_1, \ldots, p_m \colon c_m\}$ where each t_i is a term, i.e., a constant or variable, $c_i \in \mathcal{C}$ is a constraint, $p_i \in [0, 1]$ are probabilities such that $\sum_i p_i \leq 1$. This makes sure that random variables are defined consistently, together with the requirement that left-hand sides of definitions are non-unifiable. Given a substitution θ that grounds $\{t_1, \ldots, t_n\}$, this definition yields constraints on a random variable $V(t_1, \ldots, t_n)\theta$. The pair $p_i \colon c_i$ means that the prior probability that the value of this random variable satisfies the constraint c_i is at least p_i. We define the *realization* of a random variable, which formally captures this notion.

Definition 1. *Let V be a random variable in a PCLP theory, where $V \sim \{p_1 : c_1, \ldots, p_m : c_m\}$. Then, a random variable X is called the* realization *of V if for all c_i, $1 \leq i \leq m$, it holds that*

$$P(c_i[V \mapsto X]) \geq p_i$$

where $c[V \mapsto X]$ is the constraint c with V substituted by X.

Constraints are not necessarily mutually exclusive; e.g., it is possible to express that 20% of the balls in an urn are definitely blue and 80% are either blue or yellow. However, to ensure that all the probability mass can be assigned to the domain of random variable V we require that the constraints are satisfiable. To specify the constraints on random variables, each constraint domain contains at least the associated random variables, interpreted logically as constants. To make sure random variables are defined independently, a constant in a constraint of the definition of V is either V itself or it does not unify with any variable in $\mathcal{V}$. For example, we may include the constraint $\mathrm{Support}(apple) = yes$ in the definition of $\mathrm{Support}(apple)$, but not refer to $\mathrm{Support}(banana)$.

3.2 Semantics

As the theory $\mathcal{T}$ specifies constraints on random variables, typically there is a class of distributions that satisfies the constraints. In fact, for a given ground atom q, the theory specifies an interval $[P_{\min}(q), P_{\max}(q)]$ such that $P_{\min}(q) \leq P(q) \leq P_{\max}(q)$.

Given a finite set of possible grounding substitutions $\{\theta_1, \ldots, \theta_n\}$ for each variable $V(t_1, \ldots, t_m)$ from $\mathcal{V}$, we first define a probability space over a *choice function* φ which selects for each element of $V(t_1, \ldots, t_m)\theta_i$ either one of its probability-constraint pair $p \colon c\theta_i$ or it maps to $(1 - \sum_{(p_i : c_i) \in V} p_i \colon true)$ which represents the remaining probability mass. We denote the set of grounded variables by $\mathbf{V}_{\mathcal{T}}$, which represents all the random variables in $\mathcal{T}$. A particular choice is then the set:

$$\mathrm{CH}_\varphi = \{c \mid V \in \mathbf{V}_{\mathcal{T}}, \varphi(V) = (p \colon c)\} \tag{1}$$

As each random variable is independent, the probability P_φ attached to a choice φ is the product of all selected probabilities:

$$P_\varphi = \prod_{V \in \mathbf{V}_{\mathcal{T}},\, \varphi(V) = (p \colon c)} p \tag{2}$$

Each choice can be thought of as a *possible world*, although this world is described by constraints. In this world, it is possible that q is true, although this might depend on the actual values of the random variables involved. We say that $s \in \mathcal{C}$ is a *solution* for q given a particular choice φ, denoted by *solution*(s, q, φ) iff

$$\mathbf{L}_{\mathcal{T}} \cup \mathrm{CH}_\varphi \cup \{s\} \not\models \bot \tag{3}$$

$$\mathbf{L}_{\mathcal{T}} \cup \mathrm{CH}_\varphi \cup \{s\} \models q \tag{4}$$

Given a particular choice function φ, there are three possibilities: (i) the query is necessarily true in φ, i.e., *solution*$(true, q, \varphi)$ holds, denoted by *follows*(q, φ), (ii) there are no solutions or (iii) there are some constraints which imply the query, i.e., $\exists s \colon$ *solution*(s, q, φ), denoted by *possible*(q, φ).

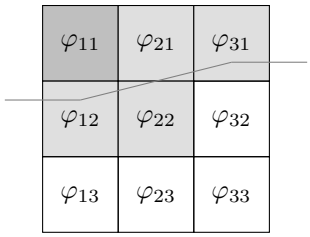

Figure 1: Two-dimensional discretized probability space with linear constraints.

Example 1. *Consider the following program:*

$\mathrm{X} \sim \{0.1 \colon 0 \leq \mathrm{X} \leq 1, 0.3 \colon 1 \leq \mathrm{X} \leq 2, 0.6 \colon 2 \leq \mathrm{X} \leq 3\}$

$\mathrm{Y} \sim \{0.1 \colon 0 \leq \mathrm{Y} \leq 1, 0.3 \colon 1 \leq \mathrm{Y} \leq 2, 0.6 \colon 2 \leq \mathrm{Y} \leq 3\}$

$q \leftarrow \mathrm{Y} < 0.75$

$q \leftarrow \mathrm{Y} < 1.25,\ 0.25\mathrm{X} + \mathrm{Y} < 1.375$

We denote the choice selecting the ith element in the definition of X *and the jth in the definition of* Y *as φ_{ij}. Those choices can be represented in a two-dimensional probability space as depicted in Figure 1. Each rectangle represents an (independent) choice of a single constraint from the distribution of two random variables; the query q is represented by the area above the line. It holds that follows(q, φ_{11}), since for this choice* $\mathrm{Y} < 1.25,\ 0.25\mathrm{X} + \mathrm{Y} < 1.375$ *always holds. In the grey areas, there is a possible solution, i.e., possible(q, φ_{ij}). For instance for the choice φ_{21},* Y *can be* 0 *and therefore less than* 0.75*. However, not all possible values consistent with the choice are a solution:* Y *can be* 1 *which is greater than* 0.75*, but less than* 1.25*.* $0.25\mathrm{X} + \mathrm{Y}$ *however can become* 1.5 *which is larger than* 1.375*.*

We can use that to define the lower bound of the probability distribution, which is the probability mass of choices for which the query is certainly true. Formally, we say the lower bound of the probability for q is the sum of the probabilities of choices which imply q, i.e.,

$$P_{\min}(q) = \sum_{\mathit{follows}(q, \varphi)} P_\varphi \tag{5}$$

For the program given in Example 1 the lower bound is $P_{\varphi_{11}} = 0.01$.

In contrast, in the upper bound we also include those choices which, only together with a particular solution, imply q, i.e.,

$$P_{\max}(q) = \sum_{\mathit{possible}(q, \varphi)} P_\varphi \tag{6}$$

For the program given in Example 1 the upper bounds is $P_{\varphi_{11}} + P_{\varphi_{21}} + P_{\varphi_{31}} + P_{\varphi_{12}} + P_{\varphi_{22}} = 0.22$.

We now introduce the concept of *solution constraints* to relate those intuitive definitions of the lower and upper bound to the possible realizations of all variables in a program.

Definition 2. *Let a solution constraint* $\mathrm{SC}(q)$ *be a constraint such that* $\mathrm{COMP}(\mathbf{R}_\mathcal{T}) \cup \mathbf{C}_\mathcal{T} \models \mathrm{SC}(q) \leftrightarrow q$*, where* $\mathrm{COMP}(\mathbf{R}_\mathcal{T})$ *is Clark's completion [Clark, 1978] of* $\mathbf{R}_\mathcal{T}$*, i.e., rules are interpreted as* if and only if *statements, so that* q *is completely characterized by* $\mathrm{SC}(q)$.

Given the solution constraint, we can now formally define the probability of q given a realization of the random variables.

Definition 3. *Given the solution constraint* $\mathrm{SC}(q)$ *having random variables* $\mathbf{V} = \{V_1, \ldots, V_n\}$ *and a set of independent random variables* $\mathbf{X} = \{X_1, \ldots, X_n\}$ *such that* X_i *is a realization of* V_i*, the probability of* q*,* $P(q \mid \mathbf{X})$*, is defined as* $P(\mathrm{SC}(q)[V_1 \mapsto X_1, \ldots, V_n \mapsto X_n])$*, denoted by* $P(\mathrm{SC}(q)[\mathbf{V} \mapsto \mathbf{X}])$.

The definition of bounds actually corresponds to the probability given the realizations for which the probability is minimal and maximal.

Theorem 1. *Given the solution constraint* $\mathrm{SC}(q)$ *and independent random variables* $\mathbf{X} = \{X_1, \ldots, X_n\}$ *such that* X_i *is a realization of* V_i*, where* $\mathbf{V} = \{V_1, \ldots, V_n\}$ *is the set of random variables which are in* $\mathrm{SC}(q)$*, it holds that* $P_{\min}(q)$ *and* $P_{\max}(q)$ *are the most tight bounds for* $P(q \mid \mathbf{X})$*, i.e.:*

$$P_{\min}(q) = \inf_{\mathbf{X}} P(q \mid \mathbf{X}) \qquad P_{\max}(q) = \sup_{\mathbf{X}} P(q \mid \mathbf{X})$$

In case the constraints only relate to a single value of the random variables, then the bounds collapse, as then *follows*$(q, \varphi) \Leftrightarrow$ *possible*(q, φ). In this case, there is a simple mapping from this logic to, e.g., *ProbLog*.

Finally, we define negations and conjunctions of queries in a similar fashion, i.e., a conjunction of literals $\mathbf{q}$ is defined by the constraint $\mathrm{SC}(\mathbf{q})$. Note that it follows that $P_{\max}(\neg q) = 1 - P_{\min}(q)$ and vice versa.

We finally define what conditional probabilities mean in this setting. For each realization of $\mathbf{V}_\mathcal{T}$ conditional probabilities are defined in the obvious way. We then define the minimum and maximum conditional probability of a program as the probability given realizations for which this probability is minimal and maximal.

Definition 4. *We define the probability bounds given evidence as:*

$$P_{\min}(q \mid e) = \inf_{\mathbf{X}} \frac{P(q \wedge e \mid \mathbf{X})}{P(e \mid \mathbf{X})}$$
$$P_{\max}(q \mid e) = \sup_{\mathbf{X}} \frac{P(q \wedge e \mid \mathbf{X})}{P(e \mid \mathbf{X})}$$

Example 2. *Recall the program introduced in Example 1 and consider an additional rule* $e \leftarrow \mathrm{X} < 1.5$*. Suppose we would like to compute* $P_{\max}(q \mid e)$*. To illustrate this probability, consider Figure 2, which is the probability space of Figure 1 with an additional constraint that implies the evidence. Intuitively, to determine* $P_{\max}(q \mid e)$ *we need to exclude choices inconsistent with* e *from the probability space and compute the probability of* q *in the remaining space. The choices* φ_{31}*,* φ_{32} *and* φ_{33} *are certainly inconsistent with* e *and are therefore excluded. The choices* φ_{21}*,* φ_{22} *and* φ_{23} *are possibly inconsistent with* e*, but only* φ_{23} *is removed. The rationale is that, since we want to maximize the probability, removing* φ_{23} *increases the resulting probability, because for all possible* $\mathbf{X}$ *it could only contribute to* $P(e \mid \mathbf{X})$ *but not to* $P(q \mid \mathbf{X})$*. In contrast, removing* φ_{21} *and* φ_{22} *would decrease the maximal probability of* q*. In summary, we compute* $P_{\max}(q \mid e) = \frac{P_{\varphi_{11}} + P_{\varphi_{21}} + P_{\varphi_{12}} + P_{\varphi_{22}}}{P_{\varphi_{11}} + P_{\varphi_{21}} + P_{\varphi_{12}} + P_{\varphi_{22}} + P_{\varphi_{13}}} \approx 0.73$*. Note that even though* $P_{\varphi_{23}}$ *would contribute to compute the upper bound of the evidence alone, it does not contribute to the partition function of* $P_{\max}(q \mid e)$.

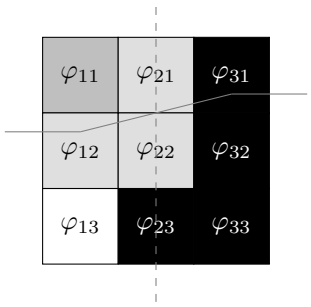

Figure 2: Two-dimensional discretized probability space with linear constraints (solid) and evidence (dashed).

To compute conditional probabilities in general we have the following proposition that relates the joint probability to the conditional probability.

Proposition 1. *The probability bounds of a query* q *given evidence* e*, as defined by Definition 4, can be computed as follows:*

$$P_{\min}(q \mid e) = \frac{P_{\min}(q \wedge e)}{P_{\min}(q \wedge e) + P_{\max}(\neg q \wedge e)}$$
$$P_{\max}(q \mid e) = \frac{P_{\max}(q \wedge e)}{P_{\max}(q \wedge e) + P_{\min}(\neg q \wedge e)}$$

4 Inference

The inference problem is solved in two steps. We first apply resolution and derive a *proof constraint* $\mathrm{PC}(q)$ for a query q, which can be seen as the operational equivalent of the solution constraint. Using this, we compute a probability for q by summing over all choices. The advantage of this two-step approach is that we only need to apply resolution once. After this, we discuss strategies to reduce the number of choices we need to sum over.

4.1 PCLP as a satisfiability problem

The collection of a proof constraint is similar to how it is done in constraint logic programming [Jaffar *et al.*, 1998] and ProbLog [Raedt *et al.*, 2007]. In brief, the idea is to apply SLD resolution on the query q, in which the constraints encountered are collected. In the end, for each proof i, we

Algorithm 1: MAXPROB

Input: proof constraint $\mathrm{PC}(q)$ and partial choice CH_φ
Result: $P_{\max}(q \mid \mathrm{CH}_\varphi)$
1 **if** $\neg\mathrm{SAT}(\ \mathrm{PC}(q) \wedge \mathrm{CH}_\varphi)$ **then return** 0
2 **if** $\neg\mathrm{SAT}(\neg\mathrm{PC}(q) \wedge \mathrm{CH}_\varphi)$ **then return** 1
3 **if** *there is some random variable* V *in* $\mathrm{PC}(q)$*, for which there is no choice in* CH_φ **then**
return $\sum_{\varphi(V)=(p:c)} p \cdot \mathrm{MAXPROB}(\mathrm{PC}(q), \mathrm{CH}_\varphi \cup \{c\})$
4
5 **else return** 1

Algorithm 2: MAXPROBDECOMP

Input: proof constraint $\mathrm{PC}(q)$
Result: $P_{\max}(q)$
1 simplify $\mathrm{PC}(q)$
2 **if** *exists independent subconstraint* C *in* $\mathrm{PC}(q)$ **then**
3 $p \quad = \mathrm{MAXPROBDECOMP}(C)$
$L| \ \ C = \mathrm{MAXPROBDECOMP}(\mathrm{PC}(q)[C \mapsto \mathit{true}])$
$L|\neg C = \mathrm{MAXPROBDECOMP}(\mathrm{PC}(q)[C \mapsto \mathit{false}])$
return $p \cdot L|C + (1-p) \cdot L|\neg C$
4 **else return** $\mathrm{MAXPROB}(\mathrm{PC}(q), \varnothing)$

obtain a conjunction of constraints $\mathrm{PC}_i(q)$ that proves q, i.e., $\mathbf{R}_\mathcal{T} \cup \{\mathrm{PC}_i(q)\} \vdash q$. Then, the complete proof constraint for q is simply the disjunction of constraints:

$$\mathrm{PC}(q) = \bigvee_{\mathbf{R}_\mathcal{T} \cup \{\mathrm{PC}_i(q)\} \vdash q} \mathrm{PC}_i(q) \tag{7}$$

Since $\mathbf{R}_\mathcal{T}$ is acyclic, there are a finite number of these proofs, though this could be generalized to cyclic theories [Mantadelis and Janssens, 2010].

Example 3. *Consider again the program from Example 1. The proof constraint for query* q *is:*

$$\mathrm{PC}(q) = \mathrm{Y} < 0.75 \vee (\mathrm{Y} < 1.25 \wedge 0.25\mathrm{X} + \mathrm{Y} < 1.375)$$

Using $\mathrm{PC}(q)$ we compute the bounds on the probability distribution by applying a satisfiability solver. We start with the probability of a single positive query atom q, which can be computed by checking the satisfiability of the proof constraint with possible choices. We denote $sat(C)$ for $\mathbf{C}_\mathcal{T} \cup \{C\} \nvdash \bot$.

Lemma 1. *For all PCLP theories* $\mathcal{T}$*, given a positive query* q*, it holds that:*

$$P_{\max}(q) = \sum_{sat(\mathrm{PC}(q) \wedge \mathrm{CH}_\varphi)} P_\varphi$$

$$P_{\min}(q) = \sum_{\neg sat(\neg\mathrm{PC}(q) \wedge \mathrm{CH}_\varphi)} P_\varphi = 1 - \sum_{sat(\neg\mathrm{PC}(q) \wedge \mathrm{CH}_\varphi)} P_\varphi$$

From this lemma, the bounds on negations and conjunctions of queries can be easily derived.

Corollary 1.

$$P_{\max}(\mathbf{q}) = \sum_{sat(\bigwedge_{q_i \in \mathbf{q}} \mathrm{PC}(q_i) \wedge \bigwedge_{\neg q_j \in \mathbf{q}} \neg\mathrm{PC}(q_j) \wedge \mathrm{CH}_\varphi)} P_\varphi$$

Example 4. *To compute* $P_{\min}(q)$ *we make use of the fact that* $P_{\min}(q) = 1 - P_{\max}(\neg q)$ *and Corollary 1. It holds that:*

$$\begin{aligned}\neg\mathrm{PC}(q) = \\ \neg(\mathrm{Y} < 0.75 \vee (\mathrm{Y} < 1.25 \wedge 0.25\mathrm{X} + \mathrm{Y} < 1.375)) = \\ \mathrm{Y} \geq 0.75 \wedge (\mathrm{Y} \geq 1.25 \vee 0.25\mathrm{X} + \mathrm{Y} \geq 1.375)\end{aligned}$$

For all φ *it holds that* $sat(\neg\mathrm{PC}(q) \wedge \mathrm{CH}_\varphi)$*, except for* φ_{11}*. Therefore,* $P_{\min}(q) = 1 - (1 - P_{\varphi_{11}}) = 0.01$.

4.2 Exact inference for PCLP

Because of the duality of $P_{\min}$ and $P_{\max}$, we will focus here on computing maximum probabilities. Clearly, by naively applying Lemma 1, it is necessary to consider all possible choices, which is exponential in the number of random variables. However, it is possible that a partial choice $\mathrm{CH}'_\varphi \subset \mathrm{CH}_\varphi$ is not consistent with or implies the proof constraint $\mathrm{PC}(q)$, which can be exploited during inference. To illustrate, let Φ_V be the possible choice functions for the random variable $\mathrm{V} \in \mathbf{V}_\mathcal{T}$ only. Then, for a proof constraint such as $\mathrm{X} > 0 \wedge \mathrm{X} + \mathrm{Y} > 0$, we naively examine $|\Phi_\mathrm{X}||\Phi_\mathrm{Y}|$ choices. However, assuming only half of the choices for X are consistent with $\mathrm{X} > 0$, we can reduce this to only $|\Phi_\mathrm{X}|/2 + |\Phi_\mathrm{X}||\Phi_\mathrm{Y}|/2$ examinations, if we consider the choices for X first. This effectively *prunes* a part of the choice space.

A basic algorithm that exploits this idea is presented in Algorithm 1. Pruning takes place at lines 1 and 2. The efficiency of the algorithm depends on the order variables are selected on line 4. As MAXPROB terminates in case an inconsistency can be found, a simple heuristic is to order random variables such that $\mathrm{V}_i < \mathrm{V}_j$ if V_i occurs in an (in)equality constraint with less variables on average than V_j. For example, in the constraint $\mathrm{X} > 0 \wedge \mathrm{X} + \mathrm{Y} > 0$ we first select choices for X as some of these choices might make the whole constraint inconsistent.

In the spirit of the well-known RelSAT algorithm [Bayardo and Pehoushek, 2000] for weighted model counting, we can also observe that in many cases the problem can be decomposed into subconstraints which do not share any random variables. For example, consider the proof constraint $\mathrm{X} > 0 \wedge \mathrm{Y} > 0$. In this case $P_{\max}(\mathrm{X} > 0 \wedge \mathrm{Y} > 0) = P_{\max}(\mathrm{X} > 0) \cdot P_{\max}(\mathrm{Y} > 0)$ which can be computed by examining $|\Phi_\mathrm{X}| + |\Phi_\mathrm{Y}|$ choices only. Also for more complex queries, independent subconstraints can be found, e.g.,

$$\begin{aligned}P_{\max}((\mathrm{X} > 0 \wedge \mathrm{Y} > 0) \vee \mathrm{X} < -3) = \\ P_{\max}(\mathrm{Y} > 0) \cdot P_{\max}(\mathrm{X} > 0 \vee \mathrm{X} < -3) + \\ (1 - P_{\max}(\mathrm{Y} > 0)) \cdot P_{\max}(\mathrm{X} < -3)\end{aligned}$$

which requires to examine $|\Phi_\mathrm{y}| + 2|\Phi_\mathrm{x}|$ choices. An algorithm that uses this idea is presented in Algorithm 2. Because of the substitutions, we initially perform some basic simplifications, e.g., in the previous example $(\mathrm{X} > 0 \wedge \mathit{false}) \vee \mathrm{X} < -3$ is simplified to $\mathrm{X} < -3$.

Theorem 2. *Given a query atom* q *and its proof constraint* $\mathrm{PC}(q)$*, then*

$$\mathrm{MAXPROBDECOMP}(\mathrm{PC}(q)) = P_{\max}(q).$$

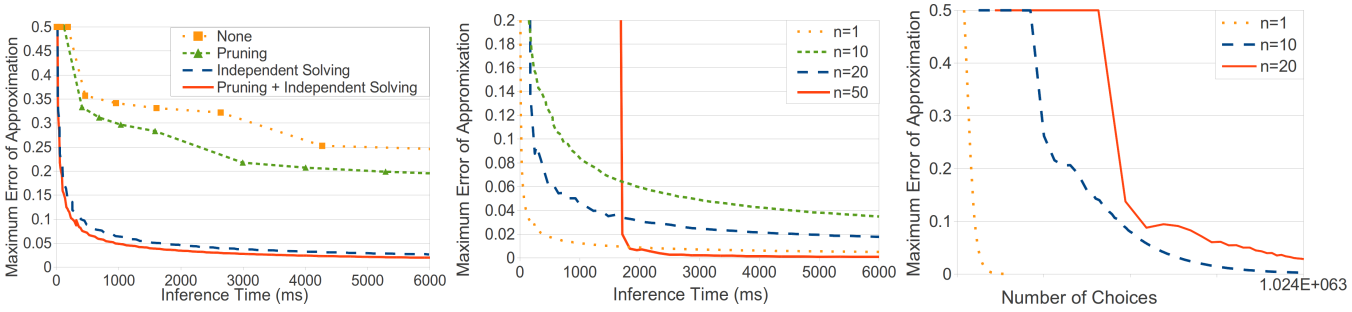

Figure 3: (a) Effect of different optimizations on inference time. (b) Scalability of inference by varying the number of queries (n) in a disjunction. (c) Number of choices needed to achieve maximum approximation error (the x-axis is logarithmic).

i.e., MAXPROBDECOMP *computes the upper bound of* q.

5 Experiments

In this section, we provide some insight into the behavior of the proposed algorithm. In particular, we investigate the proposed heuristics and provide some experiments with respect to scalability. In the implementation we make use of the satisfiability modulo theories (SMT) solver YICES [Dutertre and Moura, 2006], which supports linear arithmetic.

5.1 Heuristics

Continuing the example of Section 2, we will first present some experiments in the computation of $P(buy(apple) \vee buy(banana) \mid \text{Max\_price}(apple) < 90)$. We compare four algorithms: (i) the naive algorithm where we sum over all choices, (ii) the pruning algorithm (Algorithm 1), (iii) an algorithm that includes the decomposition of constraints (Algorithm 1), but without pruning, and (iv) the complete algorithm proposed in this paper.

Assuming we give a point prediction as the average of $\{P_{\min}(q), P_{\min}(q)\}$ then the maximum error we make is $(P_{\max}(q) - P_{\min}(q))/2$. We varied the number of intervals into which the continuous variables were discretized and plotted the relationship between the inference time and the maximum error (cf. Figure 3a).

The result shows that the proposed heuristics improve inference time significantly. In this case, identifying the independent constraints has more effect on the inference time than the pruning heuristic, as several independences could be exploited. However, we also see that the pruning of choices reduces inference time, although the effect was limited in this case: only the choices inconsistent with $\text{Max\_price}(apple) < 90$ were pruned.

5.2 Scalability

To explore how scalable our inference algorithm is we add n kinds of fruit to the program, for simplicity with the same characteristics as $banana$ and determine approximations for $P(buy(fruit_1) \vee \cdots \vee buy(fruit_n))$. The result is shown in Figure 3b where we show again the relationship between inference time and maximum error.

The result shows an interesting non-monotonic behavior: while for small n, the maximum error is higher if there are more fruits, the reverse holds for large n, e.g., if $n = 20$, then the error is smaller than for $n = 10$ given the same inference time. The first behavior occurs because the maximum error increases with the number of fruits, given the same total number of choices (see Figure 3c). So without any heuristics, the maximum error is always higher with larger n given the same inference time. However, by using the heuristics the inference time for the choice space is sub-exponential, in particular because subconstraints are independent. Therefore, the size of the choice space starts to dominate and the error can even decrease given the same inference time.

6 Conclusions & Future Work

We introduced a new *probabilistic constraint logic programs* framework that combines CLP with probabilistic inference. Although there are already several approaches where probabilistic information is combined with logic programming, many approaches are either restricted in their capabilities to representation (e.g., only to discrete distributions) or to inference (e.g., using sampling). PCLP overcomes these issues by using a constraint representation where exact inference is feasible. In general, the language allows for specifying partially unknown probability distributions.

In this paper, we also introduced an inference algorithm exploiting state-of-the-art SMT solvers, proved its correctness with respect to the semantics, introduced several heuristics, and experimentally evaluated the implementation of the inference algorithm. We obtained encouraging results indicating that inference in our language is feasible for solving challenging problems.

In future research, we aim to improve the inference further by lifting (in [Gogate and Domingos, 2011] the closely related RelSAT algorithm is lifted) and dynamically discretizing the distribution of continuous random variables [Neil *et al.*, 2007]. Finally, we aim to develop learning methods for this new language.

References

[Bacchus, 1990] F. Bacchus. *Representing and Reasoning with Probabilistic Knowlege: A Logical Approach to Probabilities*. MIT Press, Cambridge, MA, 1990.

[Bayardo and Pehoushek, 2000] Roberto J. Bayardo and Joseph Daniel Pehoushek. Counting models using connected components. In *AAAI/IAAI*, pages 157–162, 2000.

[Binder *et al.*, 1997] John Binder, Daphne Koller, Stuart Russell, Keiji Kanazawa, and Padhraic Smyth. Adaptive probabilistic networks with hidden variables. In *Machine Learning*, pages 213–244, 1997.

[Clark, 1978] K.L. Clark. Negation as failure. *Logic and Data Bases*, pages 293–322, 1978.

[Codognet and Diaz, 1996] P. Codognet and D. Diaz. Compiling Constraints in clp(FD). *J. Log. Program.*, 27(3):185–226, 1996.

[Costa and Cussens, 2003] V. Santos Costa and J. Cussens. CLP($\mathcal{BN}$): Constraint logic programming for probabilistic knowledge. In *In Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI03*, pages 517–524. Morgan Kaufmann, 2003.

[Dutertre and Moura, 2006] Bruno Dutertre and Leonardo De Moura. The yices smt solver. Technical report, Computer Science Laboratory, SRI International, 2006.

[Getoor and Taskar, 2007] L. Getoor and B. Taskar, editors. *An Introduction to Statitical Relational Learning*. MIT Press, 2007.

[Gogate and Domingos, 2011] V. Gogate and P. Domingos. Probabilistic theorem proving. In *UAI*, pages 256–265, 2011.

[Gutmann *et al.*, 2010] Bernd Gutmann, Manfred Jaeger, and Luc De Raedt. Extending problog with continuous distributions. In Paolo Frasconi and Francesca Alessandra Lisi, editors, *Proc. of the 20th International Conference on Inductive Logic Programming (ILP–10)*, Firenze, Italy, 2010.

[Gutmann *et al.*, 2011] Bernd Gutmann, Ingo Thon, Angelika Kimmig, Maurice Bruynooghe, and Luc De Raedt. The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming*, 11:663–680, 2011.

[Hentenryck, 1989] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. The MIT Press, 1989.

[Jaffar and Lassez, 1987] J. Jaffar and J.L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 111–119. ACM, 1987.

[Jaffar *et al.*, 1992] Joxan Jaffar, Spiro Michaylov, Peter J. Stuckey, and Roland H. C. Yap. The CLP(R) language and system. *ACM Trans. Program. Lang. Syst.*, 14(3):339–395, May 1992.

[Jaffar *et al.*, 1998] Joxan Jaffar, Michael Maher, Kim Marriott, and Peter Stuckey. The semantics of constraint logic programs. *The Journal of Logic Programming*, 37(13):1 – 46, 1998.

[Mantadelis and Janssens, 2010] T. Mantadelis and G. Janssens. Dedicated tabling for a probabilistic setting. In *ICLP (Technical Communications)*, pages 124–133, 2010.

[Neil *et al.*, 2007] Martin Neil, Manesh Tailor, and David Marquez. Inference in hybrid Bayesian networks using dynamic discretization. *Statistics and Computing*, 17:219–233, 2007.

[Pearl, 1988] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[Poole, 2008] David Poole. The independent choice logic and beyond. In Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors, *Probabilistic inductive logic programming*, pages 222–243. Springer-Verlag, Berlin, Heidelberg, 2008.

[Raedt *et al.*, 2007] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: a probabilistic prolog and its application in link discovery. In *Proc. of 20th International Joint Conference on Artificial Intelligence*, pages 2468–2473. AAAI Press, 2007.

[Reizler, 1998] S. Reizler. *Probabilistic Constraint Logic Programming*. PhD thesis, University of Tubingen, Tubingen, Germany, 1998.

[Sato and Kameya, 1997] Taisuke Sato and Yoshitaka Kameya. PRISM: a language for symbolic-statistical modeling. In *Proc. of the 15th International Joint Conference on Artificial Intelligence*, pages 1330–1335, 1997.

[Sato, 1995] Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *ICLP*, pages 715–729, 1995.

[Vennekens *et al.*, 2009] Joost Vennekens, Marc Denecker, and Maurice Bruynooghe. CP-logic: A Language of Causal Probabilistic Events and Its Relation to Logic Programming. *Theory and practice of logic programming*, 9:245–308, 2009.