doi:10.1017/S1471068424000437

Probabilistic Answer Set Programming with Discrete and Continuous Random Variables

DAMIANO AZZOLINI
Department of Environmental and Prevention Sciences, University of Ferrara, Ferrara, Italy
(*e-mail:* damiano.azzolini@unife.it)

FABRIZIO RIGUZZI
Department of Mathematics and Computer Science, University of Ferrara, Ferrara, Italy
(*e-mail:* fabrizio.riguzzi@unife.it)

submitted 15 March 2024; revised 19 June 2024; accepted 2 October 2024

Abstract

Probabilistic Answer Set Programming under the credal semantics extends Answer Set Programming with probabilistic facts that represent uncertain information. The probabilistic facts are discrete with Bernoulli distributions. However, several real-world scenarios require a combination of both discrete and continuous random variables. In this paper, we extend the PASP framework to support continuous random variables and propose Hybrid Probabilistic Answer Set Programming. Moreover, we discuss, implement, and assess the performance of two exact algorithms based on projected answer set enumeration and knowledge compilation and two approximate algorithms based on sampling. Empirical results, also in line with known theoretical results, show that exact inference is feasible only for small instances, but knowledge compilation has a huge positive impact on performance. Sampling allows handling larger instances but sometimes requires an increasing amount of memory.

KEYWORDS: hybrid probabilistic answer set programming, statistical relational artificial intelligence, credal semantics, algebraic model counting, exact and approximate inference

1 Introduction

Almost 30 years ago (Sato 1995), Probabilistic Logic Programming (PLP) was proposed for managing uncertain data in Logic Programming. Most of the frameworks, such as PRISM (Sato 1995) and ProbLog (De Raedt *et al.* 2007), attach a discrete distribution, typically Bernoulli or Categorical, to facts in the program and compute the success probability of a query, that is, a conjunction of ground atoms. Recently, several extensions have been proposed for dealing with continuous distributions as well (Gutmann *et al.* 2011a, 2011b; Azzolini *et al.* 2021), which greatly expand the possible application scenarios.

Answer Set Programming (ASP) (Brewka *et al.* 2011) is a powerful formalism for representing complex combinatorial domains. Most of the research in this field focuses on deterministic programs. There are three notable exceptions: LPMLN (Lee and Wang 2016), P-log (Baral *et al.* 2009), and Probabilistic Answer Set Programming (PASP) under the credal semantics (CS) (Cozman and Mauá 2016). The first assigns weights to rules, while the last two attach probability values to atoms. However, all three allow discrete distributions only.

In this paper, we extend PASP under the CS to *Hybrid* Probabilistic Answer Set Programming (HPASP) by adding the possibility of expressing continuous distributions. Our approach is based on a translation of the hybrid probabilistic answer set program into a regular probabilistic answer set program (with discrete random variables only) via a process that we call *discretization*. In this way, we can adapt and leverage already existing tools that perform inference in probabilistic answer set programs under the CS with discrete variables only. We implemented the discretization process on top of two exact solvers, based on projected answer set enumeration and knowledge compilation, respectively, and tested them on a set of five benchmarks with different configurations. Furthermore, we also implemented and tested two approximate algorithms based on sampling on the original program (with both discrete and continuous facts) and on the discretized program (with only discrete facts). Our experiments show that knowledge compilation has a huge impact in reducing the execution time and thus in increasing the scalability of the inference task. However, larger instances require approximate algorithms that perform well in almost all the tests but require, in particular, the one based on sampling the discretized program, a substantial amount of memory.

The paper is structured as follows: Section 2 introduces the needed background knowledge. Section 3 illustrates HPASP, and Section 4 presents the exact and approximate algorithms for performing inference, whose performance is tested in Section 5. Section 6 compares our proposal with related work, and Section 7 concludes the paper.

2 Background

In this section, we review the main concepts used in the paper.

2.1 *Answer set programming*

In this paper, we consider ASP (Brewka *et al.* 2011). An answer set program contains *disjunctive rules* of the form $h_1; \ldots ; h_m :- \ b_1, \ldots , b_n.$, where the disjunction of atoms h_is is called *head* and the conjunction of literals b_is is called *body*. Rules with no atoms in the head are called *constraints*, while rules with no literals in the body and a single atom in the head are called *facts*. *Choice rules* are a particular type of rules where a single head is enclosed in curly braces as in $\{a\} :- \ b_1, \ldots , b_n$, whose meaning is that a may or may not hold if the body is true. They are syntactic sugar for $a; not\_a :- \ b_1, \ldots , b_n$ where not_a is an atom for a new predicate not appearing elsewhere in the program. We allow aggregates (Alviano and Faber 2018) in the body, one of the key features of ASP that allows representing expressive relations between the objects of the domain of interest. An example of a rule $r0$ containing an aggregate is $v(A) :- \ \#count\{X : b(X)\} = A$. Here, variable A is unified with the integer resulting from the evaluation of the aggregate $\#count\{X : b(X)\}$, which counts the elements X such that $b(X)$ is true.

The semantics of ASP is based on the concept of a *stable model* (Gelfond and Lifschitz 1988). Every answer set program has zero or more stable models, also called *answer sets*. An interpretation I for a program P is a subset of its Herbrand base. The *reduct* of a program P with respect to an interpretation I is the set of rules of the grounding of P with the body true in I. An interpretation I is a stable model or, equivalently, an answer set, of P if it is a minimal model (under set inclusion) of the reduct of P w.r.t. I. We denote with $AS(P)$ the set of answer sets of an answer set program P and with $|AS(P)|$ its cardinality. If $AS(P) = \emptyset$ we say that P is unsatisfiable. If we consider rule $r0$ and two additional rules, $\{b(0)\}$ and $b(1)$, the resulting program has 2 answer sets: $\{\{b(1), b(0), v(2)\}, \{b(1), v(1)\}\}$. Finally, we will also use the concept of *projective solutions* of an answer set program P onto a set of atoms B (defined as in (Gebser *et al.* 2009)): $AS_B(P) = \{A \cap B \mid A \in AS(P)\}$.

2.2 Probabilistic answer set programming

Uncertainty in Logic Programming can be represented with discrete Boolean probabilistic facts of the form $\Pi :: a$ where $\Pi \in [0, 1]$ and a is an atom that does not appear in the head of rules. These are considered independent: this assumption may seem restrictive, but, in practice, the same expressivity of Bayesian networks can be achieved by means of rules and extra atoms (Riguzzi 2022). Every probabilistic fact corresponds to a Boolean random variable. With probabilistic facts, a normal logic program becomes a *probabilistic* logic program. One of the most widely adopted semantics in this context is the Distribution Semantics (DS) (Sato 1995). Under the DS, each of the n probabilistic facts can be included or not in a *world* w, generating 2^n worlds. Every world is a normal logic program. The DS requires that every world has a unique model. The probability of a world w is defined as $P(w) = \prod_{f_i \in w} \Pi_i \cdot \prod_{f_i \notin w} (1 - \Pi_i)$.

If we extend an answer set program with probabilistic facts, we obtain a *probabilistic* answer set program that we interpret under the CS (Cozman and Mauá 2016, 2020). In the following, when we write "probabilistic answer set program," we assume that the program follows the CS. Similarly to the DS, the CS defines a probability distribution over the worlds. However, every world (which is an answer set program in this case) may have 0 or more stable models. A query q is a conjunction of ground literals. The probability $P(q)$ of a query q lies in the range $[\underline{P}(q), \overline{P}(q)]$ where

$$\underline{P}(q) = \sum_{w_i \text{ such that } \forall m \in AS(w_i),\, m \models q} P(w_i),$$

$$\overline{P}(q) = \sum_{w_i \text{ such that } \exists m \in AS(w_i),\, m \models q} P(w_i). \tag{1}$$

That is, the lower probability is given by the sum of the probabilities of the worlds where the query is true in every answer, while the upper probability is given by the sum of the probabilities of the worlds where the query is true in at least one answer set. Here, as usual, we assume that all the probabilistic facts are independent and that they cannot

Table 1. *Worlds, probabilities, and answer sets for Example 1*

id	a	b	$P(w)$	$AS(w)$
w_0	0	0	$0.7 \cdot 0.6 = 0.42$	$\{\{\}\}$
w_1	0	1	$0.7 \cdot 0.4 = 0.28$	$\{\{q0, b\}\}$
w_2	1	0	$0.3 \cdot 0.6 = 0.18$	$\{\{a, q0\}, \{a, q1\}\}$
w_3	1	1	$0.3 \cdot 0.4 = 0.12$	$\{\{a, b, q0\}\}$

appear as heads of rules. In other words, the lower (upper) probability is the sum of the probabilities of the worlds from which the query is a cautious (brave) consequence. Conditional inference means computing upper and lower bounds for the probability of a query q given evidence e, which is usually given as a conjunction of ground literals. The formulas are by Cozman and Mauá (2017):

$$\underline{P}(q \mid e) = \frac{\underline{P}(q, e)}{\underline{P}(q, e) + \overline{P}(\neg q, e)}$$

$$\overline{P}(q \mid e) = \frac{\overline{P}(q, e)}{\overline{P}(q, e) + \underline{P}(\neg q, e)} \qquad (2)$$

Conditional probabilities are undefined if the denominator is 0.

Example 1.
The following probabilistic answer set program has two probabilistic facts, $0.3 :: a$ and $0.4 :: b$:

```
0.3::a. 0.4::b.
q0 ; q1:- a. q0:- b.
```

There are $2^2 = 4$ worlds: $w_0 = \{not\, a, not\, b\}$, $w_1 = \{not\, a, b\}$, $w_2 = \{a, not\, b\}$, and $w_3 = \{a, b\}$ (with $not\, f$ we mean that the fact f is absent, not selected), whose probability and answer sets are shown in Table 1. For example, the answer sets for w_2 are computed from the following program: $a.\, q0; q1 :\!-\, a.\, q0 :\!-\, b.$ If we consider the query $q0$, it is true in all the answer sets of w_1 and w_3 and in one of those of w_2; thus, we get $[P(w_1) + P(w_3), P(w_1) + P(w_2) + P(w_3)] = [0.4, 0.58]$ as probability bounds. □

DS and CS can be given an alternative but equivalent definition based on sampling in the following way. We repeatedly sample worlds by sampling every probabilistic fact obtaining a normal logic program or an answer set program. Then, we compute its model(s), and we verify whether the query is true in it (them). The probability of the query under DS is the fraction of worlds in whose model the query is true as the number of sampled worlds tends to infinity. For CS, the lower probability is the fraction of worlds where the query is true in all models, and the upper probability is the fraction of worlds where the query is true in at least one model. We call this the *sampling semantics,* and it is equivalent to the DS and CS definitions by the law of large numbers.

Table 2. *Tight complexity bounds of the CR problem in PASP from Mauá and Cozman (2020). Not stratified denotes programs with stratified negations and ∨ disjunction in the head*

Language	Propositional	Bounded Arity
{}	PP	PP^{NP}
{not stratified}	PP	PP^{NP}
{not}	PP^{NP}	$PP^{\Sigma_2^p}$
{∨}	PP^{NP}	$PP^{\Sigma_2^p}$
{not stratified, ∨}	$PP^{\Sigma_2^p}$	$PP^{\Sigma_3^p}$
{not, ∨}	$PP^{\Sigma_2^p}$	$PP^{\Sigma_3^p}$

The complexity of the CS has been thoroughly studied by Mauá and Cozman (2020). In particular, they focus on analyzing the cautious reasoning (CR) problem: given a PASP P, a query q, evidence e, and a value $\gamma \in \mathbb{R}$, the result is positive if $\underline{P}(q \mid e) > \gamma$, negative otherwise (or if $\underline{P}(e) = 0$). A summary of their results is shown in Table 2. For instance, in PASP with stratified negation and disjunctions in the head, the complexity of the CR problem is in the class $PP^{\Sigma_2^p}$, where PP is the class of the problems that can be solved in polynomial time by a probabilistic Turing machine (Gill 1977). The complexity is even higher if aggregates are allowed.

2.3 ProbLog and hybrid probLog

A ProbLog (De Raedt *et al.* 2007) program is composed by a set of Boolean probabilistic facts as described in Section 2.2 and a set of definite logical rules, and it is interpreted under the DS. The probability of a query q is computed as the sum of the probabilities of the worlds where the query is true: every world has a unique model, so it is a sharp probability value.

Gutmann *et al.* (2011a) proposed Hybrid ProbLog, an extension of ProbLog with continuous facts of the form

$$(X, \phi) :: b$$

where b is an atom, X is a variable appearing in b, and ϕ is a special atom indicating the continuous distribution followed by X. An example of continuous fact is $(X, gaussian(0,1)) :: a(X)$, stating that the variable X in $a(X)$ follows a Gaussian distribution with mean 0 and variance 1. A Hybrid ProbLog program P is a pair (R, T) where R is a set of rules and $T = T^c \cup T^d$ is a finite set of continuous (T^c) and discrete (T^d) probabilistic facts. The value of a continuous random variable X can be compared only with constants through special predicates: $below(X, c_0)$ and $above(X, c_0)$, with $c_0 \in \mathbb{R}$, which succeed if the value of X is, respectively, less than or greater than c_0, and $between(X, c_0, c_1)$, with $c_0, c_1 \in \mathbb{R}$, $c_0 < c_1$, which succeeds if $X \in [c_0, c_1]$.

The semantics of Hybrid ProbLog is given in Gutmann *et al.* (2011a) a proof theoretic way. A vector of samples, one for each continuous variable, defines a so-called continuous subprogram, so the joint distribution of the continuous random variables defines a joint distribution (with joint density $f(\mathbf{x})$) over continuous subprograms. An interval $I \in \mathbb{R}^n$, where n is the number of continuous facts, is defined as the cartesian product of an interval for each continuous random variable, and the probability $P(\mathbf{X} \in I)$ can be computed by integrating $f(\mathbf{x})$ over I, that is,

$$P(\mathbf{X} \in I) = \int_I f(\mathbf{x})\, d\mathbf{x}.$$

Given a query q, an interval I is called *admissible* if, for any $\mathbf{x}$ and $\mathbf{y}$ in I and for any truth value of the probabilistic facts, the truth value of the q evaluated in the program obtained by assuming $\mathbf{X}$ takes value $\mathbf{x}$ and $\mathbf{y}$ is the same. In other words, inside an admissible interval, the truth value of a query is not influenced by the values of the continuous random variables; that is, it is always true or false once the value of the discrete facts is fixed. For instance, if we have the continuous fact $(X, \mathit{gaussian}(0,1)) :: a(X)$ and a rule $q :- a(X), \mathit{between}(X, 0, 2)$, the interval $[0, 3]$ is not admissible for the query q, while $[0.5, 1.5]$ is (since for any value of $X \in [0.5, 1.5]$, $\mathit{between}(X, 0, 2)$ is always true). A partition is a set of intervals, and it is called admissible if every interval is admissible. The probability of a query is defined by extracting its proofs together with the probabilistic facts, continuous facts, and comparison predicates used in the proofs. These proofs are finitely many since the programs do not include function symbols and generalize the infinitely many proofs of a hybrid program (since there can be infinitely many values for continuous variables). Gutmann *et al.* (2011a) proved that an admissible partition exists for each query having a finite number of proofs and the probability of a query does not depend on the admissible partition chosen. They also propose an inference algorithm that first computes all the proofs for a query. Then, according to the continuous facts and comparison predicates involved, it identifies the needed partitions. To avoid counting multiple times the same contribution, the proofs are made disjoint. Lastly, the disjoint proofs are converted into a binary decision diagram (BDD) from which the probability can be computed by traversing it bottom up (i.e., with the algorithm adopted in ProbLog (De Raedt *et al.* 2007)).

Let us clarify it with a simple example.

Example 2.
The following Hybrid ProbLog program has one discrete fact and one continuous fact.

```
0.4::a.
(X,gaussian(0,1))::b(X).
q:- a.
q:- b(X), above(X,0.3).
```

Consider the query q. It has two proofs: a and $b(X), \mathit{above}(X, 0.3)$. The probability of q is computed as $P(a) \cdot P(X > 0) + P(a) \cdot (1 - P(X > 0)) + (1 - P(a)) \cdot P(X > 0) = 0.4 \cdot 0.5 + 0.4 \cdot 0.5 + 0.6 \cdot 0.5 = 0.7$. □

The Hybrid ProbLog framework imposes some restrictions on the continuous random variables and how they can be used in programs, namely: (i) comparison predicates must compare variables with numeric constants, (ii) arithmetic expressions involving continuous random variables are not allowed, and (iii) terms inside random variables definitions can be used only inside comparison predicates. These restrictions allow the algorithm for the computation of the probability to be relatively simple.

3 Hybrid probabilistic answer set programming

Probabilistic logic programs that combine both discrete and continuous random variables are usually named *hybrid*.[1] In this paper, we adopt the same adjective to describe probabilistic answer set programs with both discrete and continuous random variables, thus introducing *hybrid* probabilistic answer set programs. We use the acronym (H)PASP to indicate both (Hybrid) Probabilistic Answer Set Programming and (hybrid) probabilistic answer set program; the meaning will be clear from the context.

Without loss of generality, we consider only ground discrete and continuous probabilistic facts. With the syntax

```
f : distribution.
```

where f is ground, we indicate a continuous random variable f that follows the distribution specified by *distribution*. For example, to state that a is a continuous random variable that follows a Gaussian distribution with mean 2 and variance 1, we write

```
a : gaussian(2,1).
```

Definition 1.
A hybrid probabilistic answer set program is a triple (D, C, R), *where* D *is a finite set of ground independent probabilistic facts,* C *is a finite set of continuous random variables definitions, and* R *is a set of rules with none of the atoms in* D *or* C *in the head.* □

As in Hybrid ProbLog, we reserve some special predicates (that we call *comparison predicates*) to compare the value of the random variables with numeric ($\in \mathbb{R}$) constants: $above(Var, value)$, which is true if the value for the variable Var is greater than the numeric value $value$ ($Var > value$); $below(Var, value)$, which is true if the value for the variable Var is less than the numeric value $value$ ($Var < value$); $between(Var, l, u)$, which is true if the value for the variable Var is between the range defined by the numeric values l and u (i.e., $Var > l$ and $Var < u$); and $outside(Var, l, u)$, which is true if the value for the variable Var is outside the range defined by the numeric values l and u (i.e., $Var < l$ or $Var > u$). Given a value for a random variable, we can evaluate the atoms for the comparison predicates. For example, if variable a has value 1.2, $below(a, 1.8)$ is true, $above(a, 1.8)$ is false, $between(a, 1.1, 1.8)$ is true, and $outside(a, 1.1, 1.8)$ is false. In our framework, we consider the same restrictions of Gutmann *et al.* (2011a). Note that $outside/3$ literals are not allowed by Hybrid ProbLog.

1 In ASP terminology, the word "hybrid" is usually adopted to describe an extension of ASP, such as the one by Janhunen *et al.* (2017). Here we use the word hybrid exclusively to denote the presence of discrete and continuous random variables.

We can extend the sampling semantics for the CS to hybrid probabilistic answer set programs: we sample every probabilistic fact and a value for every continuous variable, obtaining sample $s=(d,c)$, where d is the sample for discrete facts and c is the sample for continuous random variables. From s, we build a *hybrid world* $w=(P,c)$ where P is obtained by adding to R each fact f_i for which $p_i :: f_i \in D$ and $d_i = 1$ in vector d. We then ground the rules and check the sampled values for the continuous random variables against every comparison predicate that contains the variable: (i) if the comparison is false, we remove the grounding of the rule, or (ii) if the comparison is true, we remove the comparison predicate from the body of the ground rule. We are left with an answer set program $w^{'}$, and we can compute its stable models. The lower and upper probability of a query q are given, as in the sampling semantics for CS, as the fraction of worlds where the query is true in every answer set and in at least answer set, respectively, as the number of samples goes to infinity. That is,

$$\underline{P}(q) = \lim_{n\to\infty} \frac{\sum_{i=1}^{n} \mathbb{1}(\forall m \in AS(w_i^{'}), m \models q)}{n}$$

$$\overline{P}(q) = \lim_{n\to\infty} \frac{\sum_{i=1}^{n} \mathbb{1}(\exists m \in AS(w_i^{'}), m \models q)}{n} \tag{3}$$

where $\mathbb{1}$ is the indicator function, returning 1 if its argument is true, 0 otherwise, and $AS(w_i^{'})$ is the set of answer sets of the program $w_i^{'}$ obtained from the i-th sample. We call this the *hybrid sampling semantics*.

To better specify the syntax, let us introduce an example.

Example 3.
Consider a medical domain where we are interested in computing the probability of stroke given the values of the blood pressure of some individuals.

```
0.4::pred_d(1..4).
0.6::pred_s(1..4).
d(1..4):gamma(70,1).
s(1..4):gamma(120,1).

prob_d(P):- outside(d(P), 60, 80).
prob_s(P):- outside(s(P), 110, 130).

prob(P):- prob_d(P), pred_d(P).
prob(P):- prob_s(P), pred_s(P).

stroke(P);not_stroke(P):- prob(P).

:- #count{X:prob(X)}=P,
   #count{X:stroke(X),prob(X)}=S,
   10*S < 4*P.

high_number_strokes:-
   #count{X : stroke(X)}=CS, CS > 1.
```

For ease of explanation and conciseness, we use the syntax $a(l..u)$ with $l, u \in \mathbb{N}$, $l < u$, to denote a set of atoms $a(l), a(l+1), \ldots, a(u)$. The program considers four people indexed with 1, 2, 3, and 4. The first two lines introduce eight discrete probabilistic facts, $pred\_d(i)$, $i \in \{1, \ldots, 4\}$ (pred is short for predisposition), which are true with probability 0.4, and $pred\_s(i)$, $i \in \{1, \ldots, 4\}$, which are true with probability 0.6, and eight continuous random variables ($d(1), \ldots, d(4), s(1), \ldots, s(4)$, where d stands for diastolic and s for systolic), where the first four follow a gamma distribution with shape 70 and rate 1 and while the remaining follow a gamma distribution with shape 120 and rate 1. $prob\_d/1$ indicates a problem from the diastolic pressure, and it is true if there is a problem coming from the diastolic pressure if its value is below 60 or above 80. Similarly for $prob\_s/1$. In general, a person has a blood pressure problem ($prob/1$) if they have either a diastolic or systolic pressure problem, and there is a predisposition for it (either $pred\_d$ or $pred\_s$). A person having a blood pressure problem can have a stroke or not. The constraint states that at least 40% of people that have a pressure problem also have a stroke. Finally, we may be interested in computing the probability that more than one person has a stroke ($high\_number\_strokes/0$). $\square$

The comparison predicates subdivide the domain of a random variable into disjoint and exhaustive intervals $I_1 \cup I_2 \cup \cdots \cup I_m$. The extremes of the disjoint intervals are obtained by selecting all constants appearing in comparison predicates for continuous random variables, removing the duplicates, and ordering them in increasing order. In this way, we obtain a list of values $[b_1 \ldots, b_{m+1}]$ where $b_1 = -\infty$ and $b_{m+1} = +\infty$ such that $I_k = [b_k, b_{k+1}]$ for $k = 1, \ldots, m$. This process is described in Algorithm 2 of Gutmann *et al.* (2011a).

Example 4.
Consider the following simple program:

```
0.4::b. a:gaussian(0,1).
q0 ; q1:- below(a,0.5).
q0:- below(a,0.7), b.
```

There are 3 intervals to consider: $I_1^a =]-\infty, 0.5]$, $I_2^a = [0.5, 0.7]$, and $I_3^a = [0.7, +\infty[$. $\square$

We transform a HPASP P_c into a PASP P_c^d via a process that we call *discretization*. The probability of a query q in P_c is the same as the probability asked in P_c^d. Thus, using discretization, inference in HPASP is performed using inference in PASP. We now show how to generate the discretized program, and then we prove the correctness of the transformation.

We need to make the rules involving comparison predicates considering more than one interval disjoint, to avoid counting multiple times the same contribution. To do so, we can first convert all the comparison predicates $between/3$ and $outside/3$ into a combination of $below/2$ and $above/2$. That is, every rule,

$$h :- \ l_1, \ldots, between(a, lb, ub), \ldots, l_n.$$

is converted into

$$h :- \ l_1, \ldots, above(a, lb), below(a, ub), \ldots, l_n.$$

The conversion of $outside(a, lb, ub)$ requires introducing two rules. That is,

$$h :- \ l_1, \ldots, outside(a, lb, ub), \ldots, l_n.$$

requires generating two rules

$$h_a :- \ l_1, \ldots, above(a, ub), \ldots, l_n.$$

$$h_b :- \ l_1, \ldots, below(a, lb), \ldots, l_n.$$

If there are multiple comparison predicates in the body of a rule, the conversion is applied multiple times, until we get rules with only $above/2$ and $below/2$ predicates. After this, for every continuous variable f_j with m associated intervals, we introduce a set of clauses (De Raedt *et al.* 2008; Gutmann *et al.* 2011a)

$$\begin{aligned}
&h^j_1 :- \ f_{j1}.\\
&h^j_2 :- \ not\, f_{j1}, f_{j2}.\\
&\ldots\\
&h^j_{m-1} :- \ not\, f_{j1}, not\, f_{j2}, \ldots, f_{jm-1}.\\
&h^j_m :- \ not\, f_{j1}, not\, f_{j2}, \ldots, not\, f_{jm-1}.
\end{aligned} \tag{4}$$

where each h^j_i is a propositional atom that is true if the continuous variable f_j takes value in the interval $I^{f_j}_i$ and each f_{jk} is a fresh probabilistic fact with probability π_{jk} for $k = 1, \ldots, m-1$ computed as

$$\pi_{jk} = \frac{P(b_k \leq f_j \leq b_{k+1})}{1 - P(f_j \leq b_k)} = \frac{\int_{b_k}^{b_{k+1}} p_j(x_j)\, dx_j}{1 - \int_{-\infty}^{b_k} p_j(x_j)\, dx_j} \tag{5}$$

where $p_j(x_j)$ is the probability density function of the continuous random variable f_j and b_k for $k = 1, \ldots, m$ are the bounds of the intervals. After this step, we identify the comparison atoms that are true in more than one interval. A clause containing a comparison atom $below(f_j, b_{k+1})$ is replicated k times, once for each of the intervals I_l with $l = 1, \ldots, k$. The comparison atom of the k-th replicated clause is replaced by h^j_k. Similarly for $above/2$. If a clause contains comparison atoms on multiple variables, this process is repeated multiple times. That is, if a clause contains n_c comparison atoms on the variables $v_1, \ldots, v_n$ that are true in $k_1, \ldots, k_n$ intervals, we get $k_1 \times \cdots \times k_n$ clauses. Note that the complexity of inference is dominated by the number of probabilistic facts, rather than the number of clauses.

Algorithm 1 depicts the pseudocode for the discretization process applied to a HPASP with ground rules R, continuous random variable definitions C, and discrete probabilistic facts D. It is composed of three main functions, ConvertBetweenOutside,

Algorithm 1. Function Discretize: discretization of a hybrid probabilistic answer set program with rules R, continuous probabilistic facts C and discrete probabilistic facts$D$.

```
function DISCRETIZE($R, C, D$)
    $R^d \leftarrow$ CONVERTBETWEENOUTSIDE($R$)
    $R^d, D \leftarrow$ HANDLEINTERVALS($R^d, C, D$)
    $R^d \leftarrow$ HANDLECOMPARISONATOMS($R^d$)
    return $R^d, D$
end function

function CONVERTBETWEENOUTSIDE($R$)
    $R^d \leftarrow \{\}$
    for all $r \in R$ do
        $r_c \leftarrow$ CONVERTBETWEEN($r$)
        $R^d \leftarrow R^d \cup r_c$
    end for
    while $\exists r \in R^d : outside/3 \in r$ do
        $r_a, r_b \leftarrow$ CONVERTOUTSIDE($r$)
        $R^d \leftarrow R^d \setminus \{r\} \cup \{r_a, r_b\}$
    end while
    return $R^d$
end function

function HANDLEINTERVALS($R^d, C, D$)
    for all $c \in C$ do
        $n_i \leftarrow$ COMPUTEINTERVALS($c$)  ▷ Computation of the intervals for the continuous probabilistic fact $c$.
        for $i \in \{1, \ldots, n_i\}$ do
            $f_i \leftarrow$ COMPUTEPROBABILITY($c, i$)        ▷ $f_i$ is a fresh probabilistic fact for the current interval $i$.
            $D \leftarrow D \cup f_i$
            $R^d \leftarrow R^d \cup \{h_i \leftarrow \bigwedge_{j=1}^{i-1} not\ f_j\} \land f_i$
        end for
    end for
    return $R^d, D$
end function

function HANDLECOMPARISONATOMS($R$)
    $K \leftarrow$GETCOMPARISONATOMS($R$)
    for all $a \in K$ do                                  ▷ Loop over all the comparison atoms.
        $i_c \leftarrow$ COMPUTEINTERVALSCOMPARISON($a$)
        $R^t \leftarrow \{\}$
        for all $r \in R$ do
            if $a \in R$ then
                for $i \in \{1, \ldots, i_a\}$ do    ▷ Loop over the intervals that make the comparison atom $a$ true.
                    $r_a \leftarrow$REPLACE($r, a, h_i$)
                    $R^t \leftarrow R^t \cup \{r_a\}$
                end for
            else
                $R^t \leftarrow R^t \cup \{r\}$
            end if
        end for
        $R \leftarrow R^t$
    end for
    return $R$
end function
```

HandleIntervals, and HandleComparisonAtoms that convert the between and outside comparison predicates, compute the number of intervals and the new discrete probabilistic facts, and manage the comparison atoms, respectively. Functions ConvertBetween and ConvertOutside convert, respectively, the comparison predicates $between/3$ and $outside/3$ into combinations of $above/2$ and $below/2$. Function ComputeIntervals(c) computes the intervals for the continuous random variable c.

Function ComputeProbability(c, i) computes the probability for the i-th probabilistic fact for the continuous random variable c. Function GetComparisonAtoms(R) returns all the comparison atoms in the rules R of the program. Function ComputeIntervalsComparison(c) computes the number of intervals involved in the comparison atom c. Function Replace(a, h_i) replaces the comparison atom a with the corresponding discrete fact h_i representing the i-th interval.

Theorem 1.
Consider an hybrid probabilistic answer set program P. Let n_r be the number of rules, n_c be the number of continuous facts, n_k be the number of comparison atoms in the program, n_i be the maximum number of intervals for a continuous fact, o be the maximum number of outside/3 atoms in the body of a rule, and i_c be the maximum number of intervals where a comparison atom is true. Then, Algorithm 1 requires $O(n_r^o + n_c \cdot n_i + n_k \cdot n_r \cdot i_c)$ time.

Proof.
Let us analyze the three functions used in Discretize. For function ConvertBetweenOutside, the body of the loop at lines 10–13 is executed n_r times. The while loop at lines 14–17 is executed n_r^o times. Thus, function ConvertBetweenOutside has complexity $O(n_r^o)$. Function HandleIntervals has complexity $O(n_c \cdot n_i)$. In function HandleComparisonAtoms, the loop at lines 35–49 is executed n_k times. For each of them, the loop at lines 38–47 is executed n_r times. Lastly, the innermost loop at lines 40–43 is executed at most i_c times. So, function HandleComparisonAtoms has complexity $O(n_k \cdot n_r \cdot i_c)$. Overall, function Discretize has complexity $O(n_r^o + n_c \cdot n_i + n_k \cdot n_r \cdot i_c)$. $\square$

Note that the complexity of inference in probabilistic answer set programs is very high (see Table 2), so the discretization process has a small impact on the overall process.

We can perform conditional inference using the formulas for discrete programs (equation (2)): if the evidence is on a discrete variable, we can directly apply that formula to the discretized program. If the evidence is on a continuous variable, say, $e = (X > k)$ with $k \in \mathbb{R}$, we first need to create a discretized version of the program by also taking into account this constraint.

To clarify, in Example 4, we have a variable a with two numerical constraints on it: $below(a, 0.5)$ and $below(a, 0.7)$. The first interval, I_1^a, is $]-\infty, 0.5]$, the second, I_2^a, is $[0.5, 0.7]$, and the third $[0.7, \infty[$. After inserting the new probabilistic facts, f_{a1} for I_1^a and f_{a2} for I_2^a, we add two more rules $h_1^a \mathrel{:-} f_{a1}.$ and $h_2^a \mathrel{:-} not\ f_{a1}, f_{a2}.$, we replicate the clauses with comparison predicates spanning more than one interval and replace the comparison predicates with the appropriate h_i^a atom, to make them disjoint, as previously above. For example, the comparison atom $below(a, 0.7)$ of Example 4 is true in the intervals I_1^a and I_2^a. The clause containing it is duplicated, and in the first copy, we replace $below(a, 0.7)$ with h_1^a, while in the second with h_2^a. The other comparison atom, $below(a, 0.5)$, is true in only one interval, so it is sufficient to replace it with h_1^a. This process is repeated for every continuous random variable. Overall, we obtain the program shown in Example 5.

Table 3. *Worlds and probabilities for Example 4. f_1 and f_2 are the two probabilistic facts obtained by the discretization process of the continuous probabilistic fact a into three intervals. The column LP/UP indicates whether the world contributes only to the upper probability (UP) or also to the lower probability (LP+ UP)*

id	b	f_1	f_2	LP/UP	$P(w)$
0	0	0	0	-	0.145
1	0	0	1	-	0.040
2	0	1	0	UP	0.325
3	0	1	1	UP	0.089
4	1	0	0	-	0.097
5	1	0	1	LP + UP	0.027
6	1	1	0	LP + UP	0.217
7	1	1	1	LP + UP	0.060

Example 5.
By applying the discretization process to Example 4, we get that $\pi_{a1} = 0.6915$ and $\pi_{a2} = \frac{0.066}{1-0.6915} = 0.2139$. The program becomes:

```
0.4::b.
0.6915::a1.
0.2139::a2.
int1 :- a2.
int2 :- not a1, a2.
q0 ; q1:- int1.
q0:- int1, b.
q0:- int2, b.
```

We can now compute the probability of the query, for example, $q0$: $[\underline{P}(q0), \overline{P}(q0)] = [0.303, 0.718]$. The worlds are shown in Table 3. Similarly, for the program of Example 3, we get $P(high\_number\_strokes) = [0.256, 0.331]$. □

We now show that the lower and upper probability of a query from the discretized program is the same as that from the hybrid sampling semantics.

Theorem 2.
Given a query q, a hybrid program P, and its discretized version P^d, the lower and upper probability of q computed in the hybrid program ($\underline{P}(q)$ and $\overline{P}(q)$) coincide with the lower and upper probability computed in its discretized version ($\underline{P}^{P^d}(q)$ and $\overline{P}^{P^d}(q)$), that is,

$$\underline{P}(q) = \underline{P}^{P^d}(q)$$

$$\overline{P}(q) = \overline{P}^{P^d}(q) \tag{6}$$

Proof.
Given a hybrid world w and a clause c that contains at least one comparison atom in the grounding of P, call $g_1, \dots, g_n$ the set of clauses in the grounding of P^d generated from c. There are two cases. The first is that the samples for the continuous random variables do not satisfy the constraints in the body of c. In this case, all the g_i clauses will have a false body. In the second case, there will be a single clause g_i, where all the h_i^j atoms in the body are true. These atoms can be removed, obtaining an answer set program $w^{'}$ that is the same as the one that would be obtained in the hybrid sampling semantics. It remains to prove that the resulting probability distributions over the discretized worlds are the same. We show that the probability of obtaining a program $w^{'}$ in the hybrid sampling semantics and in the sampling semantics from the discretized program is the same. We need to prove that the probability of a continuous random variable f_j of falling into the interval $I_k^{f_j}$ is the probability that in a world sampled from the discretized program atom h_k^j is true. The latter probability depends only on the probabilities of the f_{jl} facts. Thus it can be computed by observing that the f_{jl} facts are mutually independent: since the part of the program defining h_k^j is a stratified normal program, the lower and upper probabilities of h_k^j coincide and, if $-\infty, b_2, \dots, b_{m-1}, +\infty$ are the different bounds for the intervals they can be computed as:

$$\begin{aligned}
P(h_k^j) &= P(not\, f_{j1}) \cdot P(not\, f_{j2}) \cdot \dots \cdot P(not f_{jk-1}) \cdot P(f_{jk}) \\
&= \left(1 - \int_{-\infty}^{b_2} p_j(f_j)\, df_j\right) \cdot \frac{1 - \int_{b_2}^{b_3} p_j(f_j)\, df_j}{1 - \int_{-\infty}^{b_2} p_j(f_j)\, df_j} \cdot \dots \cdot \frac{\int_{b_k}^{b_{k+1}} p_j(f_j)\, df_j}{1 - \int_{-\infty}^{b_k} p_j(f_j)\, df_j} \\
&= \left(1 - \int_{-\infty}^{b_k} p_j(f_j)\, df_j\right) \cdot \frac{\int_{b_k}^{b_{k+1}} p_j(f_j)\, df_j}{1 - \int_{-\infty}^{b_k} p_j(f_j)\, df_j} \\
&= \int_{b_k}^{b_{k+1}} p_j(f_j)\, df_j
\end{aligned} \tag{7}$$

□

With our framework, it is possible to answer queries in programs where variables follow a mixture of distributions. For instance, in the following program

```
0.4::c.
a:gaussian(10,3).
b:gaussian(9,2).
q0:- c, above(a,6.0).
q0:- not c, above(b,6.0).
```

we can compute the probability of the query $q0$ ($[\underline{P}(q0), \overline{P}(q0)] = [0.923, 0.923]$). Here, if c is true, we consider a Gaussian distribution with mean 10 and variance 3, if c is false, a Gaussian distribution with mean 9 and variance 2. In other words, we consider a variable that follows the first Gaussian distribution if c is true and the second Gaussian distribution if c is false.

One of the key features of ASP is the possibility of adding logical constraints that cut some of the possible answer sets. However, when we consider a PASP (also obtained via discretization), constraints may cause a loss of probability mass due to unsatisfiable worlds since these contribute neither to the lower nor to the upper bound (equation (1)). There can be constraints involving only discrete probabilistic facts, constraints involving only comparison atoms (thus, continuous random variables), and constraints involving both. Let us discuss this last and more general case with an example.

Example 6.
Consider the program of Example 4 with the additional rule (constraint) $:- b, below(a, 0.2)$. When b is true, the value of a cannot be less than 0.2. This results in a loss of probability mass in the discretized program. The introduction of the constraint requires a more granular partition and an additional interval: $I_1^a =]-\infty, 0.2]$, $I_2^a = [0.2, 0.5]$, $I_3^a = [0.5, 0.7]$, and $I_4^a = [0.7, +\infty[$. Note that the probabilities of the discretized facts will be different from the ones of Example 4. □

When every world is satisfiable, we have that (Cozman and Mauá 2020):

$$\underline{P}(q) + \overline{P}(not\ q) = 1 \tag{8}$$

When at least one world is unsatisfiable, equation (8) does not hold anymore, but we have $\underline{P}(q) + \overline{P}(not\ q) + P(inc) = 1$, where $P(inc)$ is the probability of the unsatisfiable worlds. So we can still use the semantics but we need to provide the user also with $P(inc)$ beside $\underline{P}(q)$ and $\overline{P}(q)$. If we want to enforce equation (8), we can resort to normalization: we divide both the lower and the upper probability bounds of a query by the probability of the satisfiable worlds. That is, call

$$Z = \sum_{w_i \mid |AS(w_i)| > 0} P(w_i) \tag{9}$$

then $P^n(q) = [\underline{P}^n(q), \overline{P}^n(q)]$ with $\underline{P}^n(q) = \underline{P}(q)/Z$, $\overline{P}^n(q) = \overline{P}(q)/Z$. This approach has the disadvantage that if Z is 0 the semantics is not defined. In Example 6, the normalizing factor Z is 0.7683; thus, the probability of $q0$ now is $P(q0) = [0.093, 0.633]$. Note that the new bounds are not simply the bounds of Example 5 divided by Z since the constraint splits the domain even further, and the probabilities associated with the probabilistic facts obtained via discretization change.

4 Algorithms

In this section, we describe two exact and two approximate algorithms for performing inference in HPASP.

4.1 Exact inference

4.1.1 Modifying the PASTA solver

We first modified the PASTA solver (Azzolini *et al.* 2022),[2] by implementing the conversion of the hybrid program into a regular probabilistic answer set program (Algorithm 1). PASTA performs inference on PASP via projected answer set enumeration. In a nutshell, to compute the probability of a query (without loss of generality assuming here it is an atom), the algorithm converts each probabilistic fact into a choice rule. Then, it computes the answer sets projected on the atoms for the probabilistic facts and for the query. In this way, PASTA is able to identify the answer set pertaining to each world. For each world, there can be three possible cases: (i) a single projected answer set with the query in it, denoting that the query is true in every answer set, so this world contributes to both the lower and upper probability; (ii) a single answer set without the query in it: this world does not contribute to any probability; and (iii) two answer sets, one with the query and one without: the world contributes only to the upper probability. There is a fourth implicit and possible case: if a world is not present, this means that the ASP obtained from the PASP by fixing the selected probabilistic facts is unsatisfiable. Thus, in this case, we need to consider the normalization factor of equation (9). PASTA already handles this by keeping track of the sum of the probabilities of the computed worlds. The number of generated answer sets depends on the number of Boolean probabilistic facts and on the number of intervals for the continuous random variables, since every interval requires the introduction of a new Boolean probabilistic fact. Overall, if there are d discrete probabilistic facts and c continuous random variables, the total number of probabilistic facts (after the conversion of the intervals) becomes $T = d + \sum_{i=1}^{c}(k_i - 1)$, where k_i is the number of intervals for the i-th continuous fact. Finally, the total number of generated answer sets is bounded above by 2^{T+1}, due to the projection on the probabilistic facts. Clearly, generating an exponential number of answer sets is intractable, except for trivial domains.

Example 7 (Inference with PASTA.)
Consider the discretized program described in Example 5. It is converted into

```
{a1}.
{a2}.
{b}.
int1 :- a2.
int2 :- not a1, a2.
q0 ; q1:- int1.
q0:- int1, b.
q0:- int2, b.
```

It has 10 answer sets projected on the atoms $q0/0$ and $a0/0$, $a1/0$, and $b/0$: $AS_1 = \{\}$, $AS_2 = \{a1\}$, $AS_3 = \{a2\}$, $AS_4 = \{a2, q0\}$, $AS_5 = \{b\}$, $AS_6 = \{a1, a2\}$, $AS_7 = \{a1, b\}$, $AS_8 = \{a2, b, q0\}$, $AS_9 = \{a1, a2, q0\}$, and $AS_{10} = \{a1, a2, b, q0\}$. For example, the world

2 Code and datasets are available on GitHub at https://github.com/damianoazzolini/pasta and on zenodo at https://doi.org/10.5281/zenodo.11653976.

where only $a2$ is true is represented by the answer sets AS_3 and AS_4. The query $q0$ is present only in one of the two, so this world contributes only to the upper probability. The world where $a1$ and $a2$ are true and b is false is represented only by the answer set AS_9: the query is true in it so this world contributes to both the lower and upper probability. The world where only $a1$ is true is represented by AS_2, $q0$ is not present in it so it does not contribute to the probability. By applying similar considerations for all the worlds, it is possible to compute the probability of the query $q0$. □

4.1.2 Modifying the aspcs solver

We also added the conversion from HPASP to PASP to the aspcs solver (Azzolini and Riguzzi 2023a), built on top of the aspmc solver (Eiter *et al.* 2021; Kiesel *et al.* 2022), which performs inference on Second Level Algebraic Model Counting (2AMC) problems, an extension of AMC (Kimmig *et al.* 2017). Some of them are MAP inference (Shterionov *et al.* 2015), decision theory inference in PLP (Van den Broeck *et al.* 2010), and probabilistic inference under the smProbLog semantics (Totis *et al.* 2023). More formally, let X_{in} and X_{out} be a partition of the variables in a propositional theory Π. Consider two commutative semirings $\mathcal{R}_{in} = (R^i, \oplus^i, \otimes^i, e^i_\oplus, e^i_\otimes)$ and $\mathcal{R}_{out} = (R^o, \oplus^o, \otimes^o, e^o_\oplus, e^o_\otimes)$, connected by a transformation function $f : \mathcal{R}^i \to \mathcal{R}^o$ and two weight functions, w_{in} and w_{out}, which associate each literal to a weight (i.e., a real number). 2AMC requires computing:

$$
\begin{aligned}
2AMC(A) = {\bigoplus}^o_{I_{out}\in\sigma(X_{out})} {\bigotimes}^o_{a\in I_{out}} & w_{out}(a) \otimes^o \\
& f({\bigoplus}^i_{I_{in}\in\delta(\Pi|I_{out})} {\bigotimes}^i_{b\in I_{in}} w_{in}(b))
\end{aligned}
\tag{10}
$$

where $\sigma(X_{out})$ are the set of possible assignments to X_{out} and $\delta(\Pi \mid I_{out})$ are the set of possible assignments to the variables of Π that satisfy I_{out}. We can cast inference under the CS as a 2AMC problem (Azzolini and Riguzzi 2023a) by considering as inner semiring $\mathcal{R}_{in} = (\mathbb{N}^2, +, \cdot, (0,0), (1,1))$, where $+$ and $\cdot$ are component-wise and w_{in} is a function associating *not* q to $(0,1)$ and all the other literals to $(1,1)$, where q is the query. The first component n_u of the elements (n_u, n_l) of the semiring counts the models where the query is true, while the second component n_l counts all the models. The transformation function $f(n_u, n_l)$ returns a pair of values (f_u, f_l) such that $f_l = 1$ if $n_l = n_u$, 0 otherwise, and $f_u = 1$ if $n_u > 0$, 0 otherwise. The outer semiring $\mathcal{R}_{out} = ([0,1]^2, +, \cdot, (0,0), (1,1))$ is a double probability semiring, where there is a separate probability semiring for each component. w_{out} associates a to (p,p) and *not* a to $(1-p, 1-p)$ for every probabilistic fact $p :: a$, while it associates all the remaining literals to $(1,1)$. aspmc, and so aspcs, solves 2AMC using knowledge compilation (Darwiche and Marquis 2002) targeting NNF circuits (Darwiche 2004) where the order of the variables is guided by the treewidth of the program (Eiter *et al.* 2021). Differently from PASTA, aspcs does not support aggregates, but we can convert rules and constraints containing them into new rules and constraints without aggregates using standard techniques (Brewka *et al.* 2011).

4.2 Approximate inference

Approximate inference can be performed by using the definition of the sampling semantics and returning the results after a finite number of samples, similar to what is done for programs under the DS (Kimmig *et al.* 2011; Riguzzi 2013). It can be performed both on the discretized program and directly on the hybrid program.

4.2.1 Sampling the discretized program

In the discretized program, we can speed up the computation by storing the sampled worlds to avoid calling again the answer set solver in case a world has been already sampled. However, the number of discrete probabilistic facts obtained via the conversion is heavily dependent on the types of constraints in the program.

4.2.2 Sampling the hybrid program

Sampling the hybrid program has the advantage that it allows general numerical constraints, provided they involve only continuous random variables and constants. In this way, we directly sample the continuous random variables and directly test them against the constraints that can be composed of multiple variables and complex expressions. In fact, when constraints among random variables are considered, it is difficult to discretize the domain. Even if the samples would be always different (since the values are floating point numbers), also here we can perform caching, as in the discretized case: the constraints, when evaluated, are still associated with a Boolean value (true or false). So, we can store the values of the evaluations of each constraint: if a particular configuration has already been encountered, we retrieve its contribution to the probability, rather than calling the ASP solver.

4.2.3 Approximate algorithm description

Algorithm 2 illustrates the pseudocode for the sampling procedure: for the sampling on the discretized program, the algorithm discretizes the program by calling Discretize (Algorithm 1). Then, for a number of samples s, it samples a world by including or not every probabilistic fact into the program (function SampleWorld) according to the probability values, computes its answer sets with function ComputeAnswerSets (recall that a world is an answer set program), checks whether the query q is true in every answer set (function QueryInEveryAnswerSet) or in at least one answer set (function QueryInAtLeastOneAnswerSet), and updates the lower and upper probability bounds accordingly. At the end of the s iterations, it returns the ratio between the number of samples contributing to the lower and upper probability and the number of samples taken. The procedure is analogous (but without discretization) in the case of the sampling on the original program. A world is sampled with function SampleVariablesAndTestConstraints: it takes a sample for each continuous random variable, tests that value against each constraint specified in the program, and removes it if the test succeeds; otherwise, it removes the whole rule containing it. The remaining part of the algorithm is the same.

We now present two results regarding the complexity of the sampling algorithm. The first provides a bound on the number of samples needed to obtain an estimate of the upper (or lower) probability of a query within a certain absolute error. The second result provides a bound on the number of samples needed to obtain an estimate within a certain relative error.

Algorithm 2. Function Sampling: computation of the probability of a query q by taking s samples in a hybrid probabilistic answer set program P with rules R, continuous probabilistic facts C, and discrete probabilistic facts$D$. Variable type indicates whether the sampling of the discretized program or the original program is selected.

```
function SAMPLING($R, C, D, q, s, type$)
    if $type == discretized$ then
        $P^s \leftarrow$ DISCRETIZE($R, C, D$)
    else
        $P^s \leftarrow P$
    end if
    $i \leftarrow 0$
    while $i < s$ do
        if $type == discretized$ then
            $w \leftarrow$ SAMPLEWORLD($P^s$)
        else
            $w \leftarrow$ SAMPLEVARIABLESANDTESTCONSTRAINTS($P^s$)
        end if
        $as \leftarrow$ COMPUTEANSWERSETS($w$)
        if QUERYINEVERYANSWERSET($as, q$) then
            $lb = lb + 1$
            $ub = ub + 1$
        else if QUERYINATLEASTONEANSWERSET($as, q$) then
            $ub = ub + 1$
        end if
        $i \leftarrow i + 1$
    end while
    return $lp/s, up/s$
end function
```

Theorem 3 (Absolute Error.)
Let q be a query in a hybrid probabilistic answer set program P whose exact lower (upper) probability of success is p. Suppose the sampling algorithm takes s samples, k of which are successful, and returns an estimate $\hat{p} = \frac{k}{s}$ of the lower (upper) probability of success. Let ϵ and δ be two numbers in $[0, 1]$. Then, the probability that $\hat{p}$ is within ϵ of p is at least $1 - \delta$, that is,

$$P(p - \epsilon \leq \hat{p} \leq p + \epsilon) \geq 1 - \delta$$

if

$$s \geq \frac{\epsilon + \frac{1}{2}}{\epsilon^2 \delta}.$$

Proof.
We must prove that

$$P(p - \epsilon \leq \hat{p} \leq p + \epsilon) \geq 1 - \delta \tag{11}$$

or, equivalently, that

$$P(sp - s\epsilon \leq k \leq sp + s\epsilon) \geq 1 - \delta. \tag{12}$$

Since k is a binomially distributed random variable with number of trials s and probability of success p, we have that (Feller 1968):

$$P(k \geq r_1) \leq \frac{r_1(1-p)}{(r_1 - sp)^2} \tag{13}$$

if $r_1 \geq sp$. Moreover

$$P(k \leq r_2) \leq \frac{(s-r_2)p}{(sp - r_2)^2} \tag{14}$$

if $r_2 \leq sp$. Since $P(k \geq r_1) = 1 - P(k < r_1)$, from equation (13), we have

$$1 - P(k < r_1) \leq \frac{r_1(1-p)}{(r_1 - sp)^2}$$

$$P(k < r_1) \geq 1 - \frac{r_1(1-p)}{(r_1 - sp)^2} \tag{15}$$

if $r_1 \geq sp$.

In our case, we have $r_1 = sp + s\epsilon$ (since $r_1 \geq sp$) and $r_2 = sp - s\epsilon$ (since $r_2 \leq sp$). So

$$\begin{aligned}
&P(p - \epsilon \leq \hat{p} \leq p + \epsilon) = \\
&= P(r_2 \leq k \leq r_1) \\
&= P(k \leq r_1) - P(k < r_2) \\
&\geq 1 - \frac{r_1(1-p)}{(r_1 - sp)^2} - \frac{(s-r_2)p}{(sp-r_2)^2} \qquad \text{(Eq. 15 and 14 and since } P(x \leq v) \geq P(x < v)) \\
&= 1 - \frac{(sp + s\epsilon)(1-p)}{(s\epsilon)^2} - \frac{(s - sp + s\epsilon)p}{(s\epsilon)^2} \qquad \text{(by replacing the values of } r_1 \text{ and } r_2) \\
&= \frac{s^2\epsilon^2 - sp - s\epsilon + sp^2 + sp\epsilon - sp + sp^2 - sp\epsilon}{s^2\epsilon^2} \qquad \text{(by expanding)} \\
&= \frac{s\epsilon^2 - 2p - \epsilon + 2p^2}{s\epsilon^2} \qquad \text{(by collecting } s \text{ and simplifying)}
\end{aligned}$$

and

$$\begin{aligned}
\frac{s\epsilon^2 - 2p - \epsilon + 2p^2}{s\epsilon^2} &\geq 1 - \delta \\
s\epsilon^2 - 2p - \epsilon + 2p^2 &\geq s\epsilon^2 - s\epsilon^2\delta \\
s\epsilon^2\delta &\geq 2p + \epsilon - 2p^2 \\
s &\geq \frac{2p + \epsilon - 2p^2}{\epsilon^2\delta} = \frac{2p(1-p) + \epsilon}{\epsilon^2\delta}.
\end{aligned} \tag{16}$$

Since $0 \leq 2p(1-p) \leq \frac{1}{2}$ with $p \in [0,1]$, then equation (16) is implied by

$$s \geq \frac{\epsilon + \frac{1}{2}}{\epsilon^2\delta}$$

□

Theorem 4 (Relative Error.)
Let q be a query in a hybrid probabilistic answer set program P whose exact lower (upper) probability of success is p. Suppose the sampling algorithm takes s samples, k of which are successful, and returns an estimate $\hat{p} = \frac{k}{s}$ of the lower (upper) probability of success. Let ϵ and δ be two numbers in $[0, 1]$. Then, the probability that the error between $\hat{p}$ and p is smaller than $p\epsilon$ is at least $1 - \delta^p$, that is,

$$P(|p - \hat{p}| \leq p\epsilon) \geq 1 - \delta^p$$

if

$$s \geq \frac{3}{\epsilon^2} \ln(\frac{1}{\delta}).$$

Proof.
According to Chernoff's bound (Mitzenmacher and Upfal 2017, Corollary 4.6), we have that

$$P(|ps - k| \geq ps\epsilon) \leq 2e^{-\frac{\epsilon^2 ps}{3}}.$$

So

$$P(|p - \hat{p}| \geq p\epsilon) \leq 2e^{-\frac{\epsilon^2 ps}{3}}$$

and

$$P(|p - \hat{p}| \leq p\epsilon) \geq 1 - 2e^{-\frac{\epsilon^2 ps}{3}}.$$

Then,

$$\begin{aligned}
1 - 2e^{-\frac{\epsilon^2 ps}{3}} &\geq 1 - \delta^p \\
2e^{-\frac{\epsilon^2 ps}{3}} &\leq \delta^p \\
\ln(2) - \frac{\epsilon^2 ps}{3} &\leq p\ln(\delta) \\
-\frac{\epsilon^2 ps}{3} &\leq p\ln(\delta) - \ln(2) \leq p\ln(\delta) \\
s &\geq \frac{3}{\epsilon^2} \ln(\frac{1}{\delta}).
\end{aligned}$$

□

Please note that in Theorem 4, we exponentiate δ to p. This is needed to avoid the appearance of p in the bound since p is unknown. When p is 0, we have no guarantees on the error. When p is 1, the confidence is $1 - \delta$. For p growing from 0–1, the bound provides increasing confidence. In fact, approximating small probabilities is difficult due to the low success probability of the sample.

5 Experiments

We ran experiments on a computer with 8 GB of RAM and a time limit of 8 h (28800 s). We generated five synthetic datasets, $t1$, $t2$, $t3$, $t4$, and $t5$, where every world of all the discretized versions of the programs has at least one answer set. We use the SciPy library (Virtanen *et al.* 2020) to sample continuous random variables. The following code snippets show the PASTA-backed programs; the only difference with the ones for aspcs is in the negation symbol, *not* for the former and \+ for the latter. For all the datasets, we compare the exact algorithms and the approximate algorithms based on sampling, with an increasing number of samples. We record the time required to parse the program and to convert the HPASP into a PASP together with the inference time. The time for the first two tasks is negligible with respect to the inference time.

5.1 Dataset t1

In $t1$ we consider instances of increasing size of the program shown in Example 4. Every instance of size n has $n/2$ discrete probabilistic facts d_i, $n/2$ continuous random variables c_i with a Gaussian distribution with mean 0 and variance 1, $n/2$ pair of rules $q0 :- \mathit{below}(c_i, 0.5), \mathit{not}\, q1$ and $q1 :- \mathit{below}(c_i, 0.5), \mathit{not}\, q0$, $i = \{1, \ldots, n/2\}$, and $n/2$ rules $q0 :- \mathit{below}(c_i, 0.7), d_i$, one for each i for $i = \{1, \ldots, n/2\}$. The query is $q0$. The goal of this experiment is to analyze the behaviour of the algorithm by increasing the number of discrete and continuous random variables. The instance of size 2 is:

```
0.5::d1.
c1:gaussian(0,1).
q0 :- below(c1,0.5), not q1.
q1 :- below(c1,0.5), not q0.
q0 :- below(c1,0.7), d1.
```

5.2 Dataset t2

In $t2$ we consider a variation of $t1$, where we fix the number k of discrete probabilistic facts to 2, 5, 8, and 10, and increase the number of continuous random variables starting from 1. Every instance of size n contains k discrete probabilistic facts d_i, $i \in \{0, \ldots, k-1\}$, n continuous random variables c_i, $i \in \{1, \ldots, n\}$ with a Gaussian distribution with mean 0 and variance 1, n pair of rules $q0 :- \mathit{below}(c_i, 0.5), \mathit{not}\, q1$ and $q1 :- \mathit{below}(c_i, 0.5), \mathit{not}\, q0$, $i = \{1, \ldots, n\}$, and n rules $q0 :- \mathit{below}(c_i, 0.7), d_{(i-1)\%k}$, one for each i for $i = \{1, \ldots, n\}$. The query is $q0$. Here, when the instance size n is less than the number of discrete probabilistic facts k, $k - n$ probabilistic facts are not relevant for the computation of the probability of the query. The instance of size 2 with $k = 2$ is:

```
0.5::d0.
0.5::d1.
c1:gaussian(0,1).
c2:gaussian(0,1).
q0 :- below(c1,0.5), not q1.
q1 :- below(c1,0.5), not q0.
q0 :- below(c2,0.5), not q1.
q1 :- below(c2,0.5), not q0.
q0 :- below(c1,0.7), d0.
q0 :- below(c2,0.7), d1.
```

5.3 Dataset t3

In $t3$, we consider again a variation of $t1$ where we fix the number k of continuous random variables to 2, 5, 8, and 10, and increase the number of discrete probabilistic facts starting from 1. Every instance of size n contains k continuous probabilistic facts c_i, $i \in \{0,\dots,k-1\}$, with a Gaussian distribution with mean 0 and variance 1, n discrete probabilistic facts d_i, $i \in \{1,\dots,n\}$, k pair of rules $q0 :\!- \mathit{below}(c_i, 0.5), \mathit{not}\, q1$ and $q1 :\!- \mathit{below}(c_i, 0.5), \mathit{not}\, q0$, $i = \{1,\dots,n\}$, and n rules $q0 :\!- \mathit{below}(c_{(i-1)\%k}, 0.7), d_i$, one for each i for $i = \{1,\dots,n\}$. The query is $q0$. The instance of size 3 with $k=2$ is:

```
0.5::d1.
0.5::d2.
0.5::d3.
c0:gaussian(0,1).
c1:gaussian(0,1).
q0 :- below(c0,0.5), not q1.
q1 :- below(c0,0.5), not q0.
q0 :- below(c1,0.5), not q1.
q1 :- below(c1,0.5), not q0.
q0 :- below(c0,0.7), d1.
q0 :- below(c1,0.7), d2.
q0 :- below(c0,0.7), d3.
```

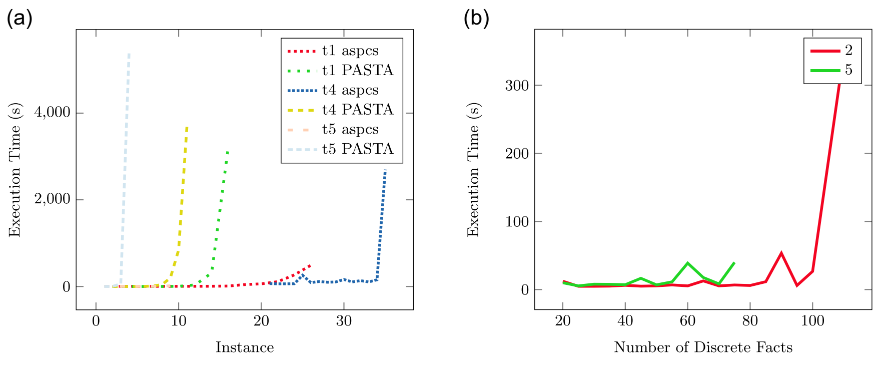
5.4 Dataset t4

In $t4$, we consider programs with one discrete probabilistic fact d and one continuous random variable a that follows a Gaussian distribution with mean 0 and standard deviation 10. An instance of size n has n pairs of rules $q0 :\!- \mathit{between}(a, lb_i, ub_i), \mathit{not}\, q1$ and $q1 :\!- \mathit{between}(a, lb_i, ub_i), \mathit{not}\, q0$ where lb_i and ub_i are randomly generated (with, $lb_i < ub_i$), for $j = 1,\dots,n$, and n rules of the form $q0 :\!- d, \mathit{between}(a, LB_j, UB_j)$, where the generation of the LB_j and UB_j follows the same process of the previous rule. We set the minimum value of lb_i and LB_j to -30 and, for both rules, the lower and upper bounds for the $\mathit{between}/2$ comparison predicate are uniformly sampled in the range $[u_{i-1}, u_{i-1} + 60/n]$, where u_{i-1} is the upper bound of the previous rule. The query is $q0$. Here, the goal is to test the algorithm with an increasing number of intervals to consider. An example of an instance of size 2 is:

```
0.4::d.
c:gaussian(0,10).
q0:- between(c,-30,-23.606), not q1.
q1:- between(c,-30,-23.606), not q0.
q0:- d, between(c,-30,-29.75).
```

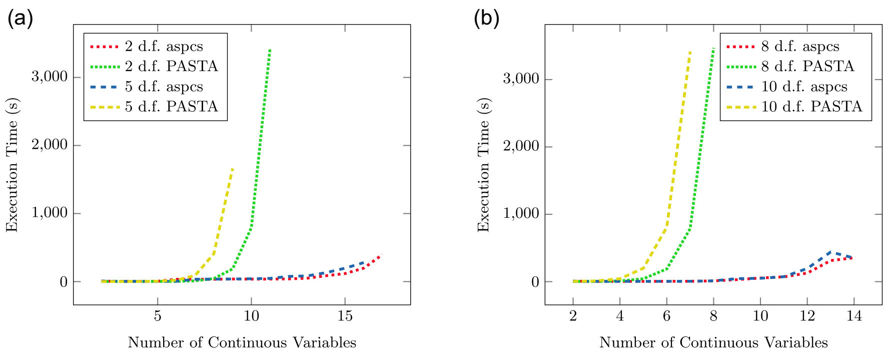
5.5 Dataset t5

Lastly, in $t5$ we consider programs of the form shown in Example 3: the instance of index n has n people involved, n discrete probabilistic facts, and n $d/1$ and $s/1$ continuous



$t1$, $t4$, and $t5$. 2 and 5 continuous facts.

Fig 1. Results for $t1$, $t4$, and $t5$ (left) and results for aspcs applied to $t3$ (right) with two and five continuous facts.



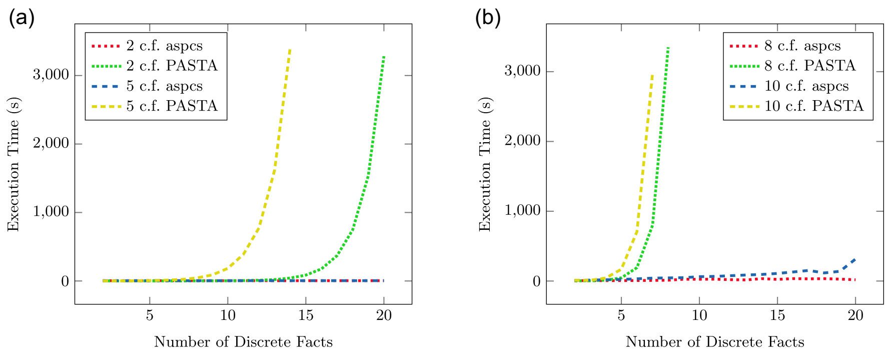
2 and 5 discrete facts. 8 and 10 discrete facts.

Fig 2. Results for the experiment $t2$ with a fixed number of discrete facts and an increasing number of continuous variables.

random variables. The remaining part of the program is the same. Example 3 shows the instance of index 4 (four people involved). The query is $high\_number\_strokes$.

5.6 Exact inference results

The goal of benchmarking the exact algorithms is threefold: (i) identifying how the number of continuous and discrete probabilistic facts influences the execution time, (ii) assessing the impact on the execution time of an increasing number of intervals, and (iii) comparing knowledge compilation with projected answer set enumeration. Figure 1(a) shows the results of exact inference for the algorithms backed by PASTA (dashed lines) and aspcs (straight lines) on $t1$, $t4$, and $t5$. For $t5$ PASTA is able to solve up to instance 4, while aspcs can solve up to instance 9 (in a few seconds). Similarly for $t1$, where aspcs doubles the maximum sizes solvable by PASTA. The difference is even more marked for



2 and 5 continuous facts. 8 and 10 continuous facts.

Fig 3. Results for the experiment $t3$ with a fixed number of continuous random variables and an increasing number of discrete facts. The x axis of the left plot is cut at size 20, to keep the results readable.

$t4$: here, PASTA solves up to instance 11 while aspcs up to instance 35. Figures 2 and 3 show the results for $t2$ and $t3$. Here, PASTA cannot solve instances with around more than 20 probabilistic facts and continuous random variables combined. The performance of aspcs are clearly superior, in particular for $t3$ with 2 and 5 continuous random variables (Figure 3(a)). Note that this is the only plot where we cut the x axis at instance 20, to keep the results readable. To better investigate the behavior of aspcs in this case, we keep increasing the number of discrete facts until we get a memory error, while the number of continuous variables is fixed to 2 and 5. Figure 1(b) shows the results: with 2 continuous variables, we can solve up to instance 110, while with 5 continuous variables, we can only solve up to 75. In general, the results of exact inference are in accordance with the theoretical complexity results. However, as also empirically shown by Azzolini and Riguzzi (2023a), knowledge compilation has a huge impact on the execution times. Overall, as expected, PASTA is slower in all the tests, being based on (projected) answer set enumeration. For all, both PASTA (exact algorithm) and aspcs stop for lack of memory.

5.7 Approximate inference results

The goal of the experiments run with approximate algorithms is: (i) analyzing the impact on the execution time of the number of samples taken, (ii) comparing the approach based on sampling the original program against the one based on sampling the converted program in terms of execution times, and (iii) assessing the memory requirements. Table 4 shows the averages on five runs of the execution time of the approximate algorithm applied to both the discretized and original program with 10^2, 10^3, 10^4, 10^5, and 10^6 samples, for each of the five datasets. Standard deviations are listed in Table 5. For $t2$ and $t3$, we consider programs with the same number of continuous variables and discrete probabilistic facts. In four of the five tests (all except $t4$), sampling the original program is slower than sampling the converted program, sometimes by a significant amount. This is probably due to the fact that sampling a continuous distribution is slower than sampling a Boolean random variable. For example, in instance 100 of $t3$, sampling the original

Table 4. *Results for the approximate algorithms based on sampling. The columns contain, from the left, the name of the dataset, the instance index, and the average time required to take* 10^2, 10^3, 10^4, 10^5, *and* 10^6 *samples on the original and converted program. O.O.M. and T.O. stand, respectively, for out of memory and timeout*

Data.	Inst.	10^2 s. O./C.	10^3 s. O./C.	10^4 s. O./C.	10^5 s. O./C.	10^6 s. O./C.
t1	50	2.81/2.21	12.46/2.07	116.07/10.19	1200.19/90.83	11998.92/906.13
t1	60	3.05/2.13	14.50/2.10	139.83/11.19	1417.36/103.63	14154.23/1059.09
t1	70	3.80/2.16	17.05/2.25	167.39/12.69	1645.97/117.31	16523.49/1222.05
t1	80	4.00/1.74	19.26/2.42	190.23/14.29	1859.69/135.09	18524.07/1375.32
t1	90	4.26/1.89	21.44/2.63	215.25/16.03	2060.15/152.52	20518.89/1558.51
t1	100	4.51/1.96	23.69/2.90	231.85/17.95	2254.84/170.81	22692.96/1751.96
t2	50	3.77/1.35	22.80/2.82	223.95/17.21	2299.33/163.13	22277.69/1654.94
t2	60	4.22/1.33	27.02/3.09	269.97/20.14	2752.39/194.99	26532.31/1986.42
t2	70	4.63/1.42	31.49/3.43	312.66/23.54	3134.76/226.24	T.O./2271.90
t2	80	4.65/1.6	36.00/3.88	363.20/27.02	3616.07/261.18	T.O./2646.70
t2	90	5.42/1.66	42.78/4.24	418.04/30.26	4006.67/293.14	T.O./2926.46
t2	100	5.77/2.18	46.35/4.63	457.80/34.05	4520.69/327.29	T.O./3286.04
t3	50	3.90/1.82	25.46/3.18	245.32/19.08	2439.00/176.84	21818.69/1808.43
t3	60	4.71/1.89	29.50/3.59	288.98/22.79	2817.13/215.89	26084.07/2176.02
t3	70	5.17/1.86	34.16/3.83	335.37/25.34	3247.42/241.67	T.O./2452.97
t3	80	5.51/2.03	38.71/4.36	382.96/29.40	3685.38/281.14	T.O./2823.65
t3	90	5.67/2.09	44.20/4.71	432.43/32.85	4123.31/318.48	T.O./3154.16
t3	100	6.15/2.15	48.83/5.17	476.51/36.47	4535.84/357.23	T.O./3542.78
t4	50	2.37/22.55	2.10/23.99	7.27/34.63	59.20/95.59	571.27/428.92
t4	60	2.42/43.08	2.29/44.18	7.81/61.99	62.36/178.80	602.12/842.64
t4	70	2.41/O.O.M.	2.35/O.O.M.	8.28/O.O.M.	66.26/O.O.M.	636.83/O.O.M.
t4	80	2.47/O.O.M.	2.61/O.O.M.	8.83/O.O.M.	69.69/O.O.M.	668.62/O.O.M.
t4	90	2.03/O.O.M.	2.89/O.O.M.	9.79/O.O.M.	75.85/O.O.M.	717.39/O.O.M.
t4	100	2.52/O.O.M.	3.09/O.O.M.	10.11/O.O.M.	77.70/O.O.M.	736.59/O.O.M.
t5	50	6.14/2.29	44.20/6.72	440.02/69.44	4433.25/1065.66	T.O./24376.32
t5	60	7.03/3.16	53.01/8.30	533.38/87.33	5437.58/1575.33	T.O./23889.43
t5	70	8.16/3.14	62.37/10.44	637.10/125.86	6497.07/1874.05	T.O./T.O.
t5	80	8.82/3.26	70.79/12.62	728.98/163.81	7610.59/2855.01	T.O./T.O.
t5	90	9.41/3.34	80.89/14.43	845.18/209.66	8931.12/3879.62	T.O./T.O.
t5	100	10.26/4.01	89.73/16.35	936.90/248.00	9984.17/4310.61	T.O./T.O.

program is over 12 times slower than sampling the converted program. However, the discretization process increases the number of probabilistic facts and so the required memory: for $t4$, from the instance 70, taking even 100 samples requires more than 8 GB of RAM. To better assess the memory requirements, we repeat the experiments with the approximate algorithms with 6, 4, 2, and 1 GB of maximum available memory. For all, taking up to 10^5 samples in the original program is feasible also with only 1 GB of memory. The same considerations hold for sampling the converted program, except for $t4$. Table 6 shows the largest solvable instances together with the number of probabilistic facts, rules (for the converted program), and samples for each instance: for example, with 1 GB of

Table 5. *Standard deviations for the results listed in Table 4. A dash denotes that there are no results for that particular instance due to either a memory error or a time limit*

Data.	Inst.	10^2 s. O./C.	10^3 s. O./C.	10^4 s. O./C.	10^5 s. O./C.	10^6 s. O./C.
t1	50	0.97/1.34	0.70/0.15	6.71/0.71	71.46/5.84	920.82/54.93
t1	60	0.99/1.19	0.76/0.11	9.10/0.65	95.47/5.71	1003.31/50.87
t1	70	1.29/1.17	0.67/0.12	10.55/0.73	86.36/7.61	1141.45/51.24
t1	80	1.24/0.92	1.01/0.10	9.52/0.65	71.22/7.98	1381.12/64.04
t1	90	1.24/1.01	1.29/0.11	10.65/0.71	56.47/7.00	1525.12/120.35
t1	100	1.20/0.96	1.37/0.24	12.62/0.92	59.29/10.79	1754.89/99.47
t2	50	0.92/0.23	0.68/0.22	11.56/0.51	150.47/3.92	438.66/39.72
t2	60	0.86/0.03	1.03/0.06	15.06/0.40	162.91/3.81	581.83/53.66
t2	70	0.93/0.11	0.86/0.10	20.00/0.44	158.68/7.14	-/68.62
t2	80	0.17/0.19	0.82/0.20	22.24/1.22	202.95/9.73	-/50.98
t2	90	0.47/0.19	3.66/0.30	30.78/1.43	218.51/12.05	-/53.71
t2	100	0.37/0.91	3.03/0.34	22.13/2.67	159.32/13.80	-/60.19
t3	50	0.07/0.11	0.85/0.11	5.92/0.59	75.95/7.63	324.79/56.21
t3	60	0.57/0.14	1.81/0.14	12.05/1.58	140.43/10.00	262.68/63.53
t3	70	0.54/0.20	2.09/0.24	15.22/1.75	162.67/12.74	-/88.86
t3	80	0.56/0.10	2.31/0.36	15.50/2.30	183.78/9.30	-/67.44
t3	90	0.20/0.13	1.43/0.34	22.31/2.25	173.10/15.10	-/100.41
t3	100	0.26/0.12	1.89/0.39	22.30/2.61	158.54/8.47	-/93.45
t4	50	1.22/0.31	0.15/1.52	0.27/3.66	2.78/6.86	6.35/13.43
t4	60	1.19/1.22	0.17/3.23	0.35/4.77	3.08/16.63	12.71/27.04
t4	70	1.21/-	0.15/-	0.35/-	4.22/-	18.99/-
t4	80	1.19/-	0.17/-	0.43/-	4.27/-	25.56/-
t4	90	0.93/-	0.15/-	0.44/-	3.42/-	27.86/-
t4	100	1.10/-	0.18/-	0.66/-	4.61/-	23.38/-
t5	50	0/0.92	1.92/0.51	17.83/8.27	122.23/208.32	-/3971.11
t5	60	0.93/1.56	2.01/0.66	22.59/10.41	177.61/238.23	-/7447.05
t5	70	1.10/1.46	2.46/0.93	20.09/14.83	158.03/400.35	-/-
t5	80	0.92/0.88	2.97/1.27	34.17/26.08	303.35/709.13	-/-
t5	90	0.82/0.94	2.15/1.37	37.42/28.92	442.68/684.92	-/-
t5	100	0.92/1.36	3.81/1.17	47.29/26.12	793.68/745.80	-/-

memory, it is possible to take only up to 10^4 samples in the instance of size 40. For this dataset, the increasing number of *between*/3 predicates requires an increasing number of rules and probabilistic facts to be included in the program during the conversion, to properly handle all the intervals: the instance of size 70 has 142 probabilistic facts and more than 30,000 rules, which make the generation of the answer sets very expensive and sampling it with only 8 GB of memory is not feasible.

6 Related Work

Probabilistic logic programs with discrete and continuous random variables have been the subject of various works. Gutmann et al. (2011a) proposed Hybrid ProbLog, an extension of ProbLog (De Raedt *et al.* 2007) to support continuous distributions. There are

Table 6. *Largest solvable instances of t4 by sampling the converted program when reducing the available memory. The columns contain, from the left, the maximum amount of memory, the largest solvable instance together with the number of probabilistic facts (# p.f.) and rules (# rules) obtained via the conversion, and the maximum number of samples that can be taken (max. # samples). Note that we increase the number of samples by starting from* 10^2 *and iteratively multiplying the number by 10, up to* 10^6*, so in the last column, we report a range: this means that we get a memory error with the upper bound while we can take the number of samples in the lower bound. Thus, the maximum values of samples lie in the specified range*

Memory	Instance	# p.f.	# rules	Max. # samples
6 GB	60	122	22495	$> 10^6$
4 GB	50	102	16151	$[10^5, 10^6]$
2 GB	50	102	16151	$[10^3, 10^4]$
1 GB	40	82	10554	$[10^4, 10^5]$

several differences with Hybrid ProbLog: first, Hybrid ProbLog focuses on PLP while our approach on PASP. Thus, the syntax and semantics are different. For the syntax, in PASP, we can use rich constructs such as aggregates that greatly increase the expressivity of the programs. For the semantics, at high level, PLP requires that every world (i.e., combination of probabilistic facts) has exactly one model while PASP does not. Another difference with Hybrid ProbLog is in the discretization process: Hybrid ProbLog discretizes the proofs of a program, while we directly discretize the program. Moreover, their inference algorithm is based on the construction of a compact representation of the program via BDDs, while we use both ASP solvers with projective solutions and knowledge compilation targeting NNF.

Distributional Clauses (Gutmann *et al.* 2011b) and Extended PRISM (Islam *et al.* 2012) are two other proposals to handle both discrete and continuous random variables. The semantics of the former is based on a stochastic extension of the immediate consequence T_p operator, while the latter considers the least model semantics of constraint logic programs (Jaffar and Maher 1994) and extends the PRISM framework (Sato 1995). Michels *et al.* (2015) introduced Probabilistic Constraint Logic Programming whose semantics is based on an extension of Sato's DS (Sato 1995). Azzolini *et al.* (2021) proposed a semantics for hybrid probabilistic logic programs that allows a denumerable number of random variables. In general, all the previously discussed approaches only support normal clauses, do not adopt some of the ASP constructs, such as aggregates and constraints, and require that the worlds have a single model. Some languages allow handling uncertainty in ASP, such as LPMLN (Lee and Wang 2016), P-log (Baral *et al.* 2009), PrASP (Nickles and Mileo 2015), and differentiable SAT/ASP (Nickles 2018) but none of these consider continuous distributions. PASOCS (Tuckey *et al.* 2021) is a system for performing inference in probabilistic answer set programs under the CS, but it does not allow worlds without answer sets and continuous variables while plingo (Hahn *et* *al.* 2022) considers the LPMLN, P-log, and ProbLog semantics (the relationship among these has been discussed in (Lee and Yang 2017)). The credal least undefined semantics (Rocha and Gagliardi Cozman 2022) and the smProbLog semantics (Totis *et al.* 2023) handle unsatisfiable worlds, but by considering three-valued semantics and do not allow continuous random variables.

There is a large body of work on inference in Probabilistic Programming (PP) (Pfeffer 2016; Gehr *et al.* 2016; Tran *et al.* 2017; van de Meent *et al.* 2021) with both discrete and continuous random variables, with several available tools (Tran *et al.* 2016; Bingham *et al.* 2018; Phan *et al.* 2019). PLP and PASP adopt a declarative approach to describe a domain, so they are particularly suitable for describing relational domains. Translating a PLP/PASP into PP is possible but would result in a much longer and less understandable program.

7 Conclusions

In this paper, we propose *H*PASP, an extension of PASP under the CS that allows both discrete and continuous random variables. We restrict the types of possible numerical constraints and, to perform exact inference, we convert the program containing both discrete probabilistic facts and continuous random variables into a program containing only discrete probabilistic facts, similarly to (Gutmann *et al.* 2011a). We leverage two existing tools for exact inference, one based on projected answer set enumeration and one based on knowledge compilation. We also consider approximate inference by sampling either the discretized or the original program. We tested the four algorithms on different datasets. The results show that the exact algorithm based on projected answer set enumeration is feasible only for small instances while the one based on knowledge compilation can scale to larger programs. Approximate algorithms can handle larger instances and sampling the discretized program is often faster than sampling the original program. However, this has a cost in terms of required memory, since the discretization process adds a consistent number of rules and probabilistic facts. In the future, we plan to extend our framework to also consider comparisons involving more than one continuous random variable and general expressions, as well as considering lifted inference approaches (Azzolini and Riguzzi 2023b) and handle the inference problem with approximate answer set counting (Kabir *et al.* 2022).

Acknowledgments

This work has been partially supported by Spoke 1 “FutureHPC & BigData” of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing funded by MUR Missione 4 – Next Generation EU (NGEU) and by Partenariato Esteso PE00000013, “FAIR – Future Artificial Intelligence Research,” Spoke 8 “Pervasive AI,” funded by MUR through PNRR – M4C2 – Investimento 1.3 (Decreto Direttoriale MUR n. 341 of March 15, 2022) under the NGEU. Both authors are members of the Gruppo Nazionale Calcolo Scientifico – Istituto Nazionale di Alta Matematica (GNCS-INdAM). We acknowledge the CINECA award under the ISCRA initiative for the availability of high-performance computing resources and support.

Competing interests

The authors declare none.

References

Alviano, M. and Faber, W. 2018. Aggregates in answer set programming. *KI-Künstliche Intelligenz* 32, 2, 119–124.

Azzolini, D., Bellodi, E. and Riguzzi, F. 2022. Statistical statements in probabilistic logic programming. In *Logic Programming and Nonmonotonic Reasoning*, G. Gottlob, D. Inclezan and M. Maratea, Eds. Cham, Springer International Publishing, 43–55.

Azzolini, D. and Riguzzi, F. 2023a. Inference in probabilistic answer set programming under the credal semantics. In *AIxIA. 2023 - Advances in Artificial Intelligence*, R. Basili, D. Lembo, C. Limongelli and A. Orlandini, Eds. vol. 14318. Lecture Notes in Artificial Intelligence. Heidelberg, Germany, Springer, 367–380.

Azzolini, D. and Riguzzi, F. 2023b. Lifted inference for statistical statements in probabilistic answer set programming. *International Journal of Approximate Reasoning* 163, 109040.

Azzolini, D., Riguzzi, F. and Lamma, E. 2021. A semantics for hybrid probabilistic logic programs with function symbols. *Artificial Intelligence*, 294, 103452.

Baral, C., Gelfond, M. and Rushton, N. 2009. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming* 9, 1, 57–144.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P. and Goodman, N. D. 2018. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research* 20, 1, 973–978.

Brewka, G., Eiter, T. and Trusczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54, 12, 92–103.

Cozman, F. G. and Maua, D. D. 2016. *The Structure and Complexity of Credal Semantics*, vol. 1661. *CEUR Workshop Proceedings*, 3–14.CEUR-WS.org.

Cozman, F. G. and Maua, D. D. 2017. On the semantics and complexity of probabilistic logic programs. *Journal of Artificial Intelligence Research*, 60, 221–262.

Cozman, F. G. and Maua, D. D. (2020). The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference. *International Journal of Approximate Reasoning*, 125, 218–239.

Darwiche, A. 2004. *New Advances in Compiling CNF into Decomposable Negation Normal Form*. IOS Press, 328–332.

Darwiche, A. and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17, 229–264.

De RAEDT, L., Demoen, B., Fierens, D., Gutmann, B., Janssens, G., Kimmig, A., Landwehr, N., Mantadelis, T., Meert, W., Rocha, R., Costa, V., Thon, I. and Vennekens, J. 2008. Towards digesting the alphabet-soup of statistical relational learning. In *NIPS. 2008 Workshop on Probabilistic Programming*.

De RAEDT, L., Kimmig, A. and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, vol. 7, AAAI Press, 2462–2467.

Eiter, T., Hecher, M. and Kiesel, R. 2021. Treewidth-aware cycle breaking for algebraic answer set counting. In M. Bienvenu, G. Lakemeyer and E. Erdem, Eds. *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021*, 269–279.

Feller, W. 1968. *An Introduction to Probability Theory and Its Applications*: Vol. 1.Wiley Series in Probability and Mathematical Statistics. John Wiley & sons.

Gebser, M., Kaufmann, B. and Schaub, T.2009.Solution enumeration for projected boolean search problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, W.-J. Van Hoeve and J. N. Hooker, Eds. Berlin, Heidelberg. Springer Berlin Heidelberg, 71–86.

Gehr, T., Misailovic, S. and Vechev, M. 2016. PSI: Exact symbolic inference for probabilistic programs. In *28th International Conference on Computer Aided Verification (CAV 2016)*, Part I, vol. 9779, S. Chaudhuri and A. Farzan, Eds. 62–83.

Gelfond, M. and Lifschitz, V.1988, The stable model semantics for logic programming. In *Proceedings of International Logic Programming Conference and Symposium*, vol. 88, USA, 1070–1080.

Gill, J. 1977. Computational complexity of probabilistic turing machines. *SIAM Journal on Computing* 6, 4, 675–695.

Gutmann, B., Jaeger, M. and De Raedt, L. 2011a. Extending problog with continuous distributions. In *Inductive Logic Programming, ILP 2010*, vol. 6489, 76–91.

Gutmann, B., Thon, I., Kimmig, A., Bruynooghe, M. and De RAEDT, L. 2011b. The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming* 11, 4-5, 663–680.

Hahn, S., Janhunen, T., Kaminski, R., Romero, J., Ruhling, N. and Schaub, T. 2022. plingo: A system for probabilistic reasoning in clingo based on LPMLN. In *Rules and Reasoning*, G. Governatori and A.-Y. Turhan, Eds. Cham: Springer International Publishing, 54–62.

Islam, M. A., Ramakrishnan, C. and Ramakrishnan, I. 2012. Inference in probabilistic logic programs with continuous random variables. *Theory and Practice of Logic Programming* 12, 4-5, 505–523.

Jaffar, J. and Maher, M. J. (1994). Constraint logic programming: A survey. *The Journal of Logic Programming* 19-20, 503–581.

Janhunen, T., Kaminski, R., Ostrowski, M., Schellhorn, S., Wanko, P. and Schaub, T. 2017. Clingo goes linear constraints over reals and integers. *Theory and Practice of Logic Programming* 17, 5-6, 872–888.

Kabir, M., Everado, F., Shukla, A., Hecher, M., Fichte, J. K. and Meel, K. S. 2022. ApproxASP - A scalable approximate answer set counter. In *AAAI Conference on Artificial Intelligence*.

Kiesel, R., Totis, P. and Kimmig, A. 2022. Efficient knowledge compilation beyond weighted model counting. *Theory and Practice of Logic Programming* 22, 4, 505–522.

Kimmig, A., Demoen, B., De Raedt, L., Costa, V. S. and Rocha, R. 2011. On the implementation of the probabilistic logic programming language robLog. *Theory and Practice of Logic Programming* 11, 2-3, 235–262.

Kimmig, A., Van Den Broeck, G. and De Raedt, L. 2017. Algebraic model counting. *Journal of Applied Logic* 22, 46–62.

Lee, J. and Wang, Y. 2016. Weighted rules under the stable model semantics. In C. Baral, J. P. Delgrande and F. Wolter, Eds. *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*. AAAI Press, 145–154.

Lee, J. and Yang, Z. 2017. LPMLN, weak constraints, and P-log. In S. Singh and S. Markovitch, Eds. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, February 4-9, 2017. AAAI Press, San Francisco, California, USA, 1170–1177.

Maua, D. D. and Cozman, F. G. 2020. Complexity results for probabilistic answer set programming. *International Journal of Approximate Reasoning* 118, 133–154.

Michels, S., Hommersom, A., Lucas, P. J. F. and Velikova, M. 2015. A new probabilistic constraint logic programming language based on a generalised distribution semantics. *Artificial Intelligence* 228, 1–44.

MITZENMACHER, M. AND UPFAL, E. 2017. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press.

NICKELS, M. 2018. Differentiable SAT/ASP. In E. BELLODI and T. SCHRIJVERS, Eds. *Proceedings of the 5th International Workshop on Probabilistic Logic Programming, PLP. 2018, co-located with the 28th International Conference on Inductive Logic Programming (ILP 2018)*, Ferrara, Italy, 1 Sept. 2018, vol. 2219 of CEUR Workshop Proceedings, 62–74.CEUR-WS.org.

NICKLES, M. AND MILEO, A. 2015. A hybrid approach to inference in probabilistic non-monotonic logic programming. In F. RIGUZZI and J. VENNEKENS, Eds. *Proceedings of the 2nd International Workshop on Probabilistic Logic Programming co-located with 31st International Conference on Logic Programming (ICLP 2015)*, Cork, Ireland, 31st Aug. 2015, vol. 1413 of CEUR Workshop Proceedings, 57–68. CEUR-WS.org.

PFEFFER, A. 2016. *Practical Probabilistic Programming*. Manning Publications.

PHAN, D., PRADHAN, N. AND JANKOWIAK, M. 2019. Composable effects for flexible and accelerated probabilistic programming in NumPyro. arXiv preprint arXiv:1912.11554.

RIGUZZI, F. 2013. MCINTYRE: A Monte Carlo system for probabilistic logic programming. *Fundamenta Informaticae* 124, 4, 521–541.

RIGUZZI, F. 2022. *Foundations of probabilistic logic programming languages, semantics, inference and learning*, 2nd edn. Gistrup, Denmark: River Publishers.

ROCHA, V. H. N. AND GAGLIARDI COZMAN, F. 2022. A credal least undefined stable semantics for probabilistic logic programs and probabilistic argumentation. In G. KERN-ISBERNER, G. LAKEMEYER and T. MEYER, Eds. *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022*, 309–319.

SATO, T.1995. A statistical learning method for logic programs with distribution semantics. In L. Sterling, Ed. *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming*, Tokyo, Japan, June 13-16, 1995, MIT Press, 715–729.

SHTERIONOV, D. S., RENKENS, J., VLASSELAER, J., KIMMIG, A., MEERT, W. AND JANSSENS, G.2015. The most probable explanation for probabilistic logic programs with annotated disjunctions. In *Inductive Logic Programming*, Berlin, Heidelberg, 139–153.

TOTIS, P., DE RAEDT, L. AND KIMMIG, A. 2023. mProbLog: Stable model semantics in problog for probabilistic argumentation. *Theory and Practice of Logic Programming*, 1–50. doi: 10.1017/S147106842300008X

TRAN, D., HOFFMAN, M. D., SAUROUS, R. A., BREVDO, E., MURPHY, K. AND BLEI, D. M. 2017. Deep probabilistic programming. CoRR, abs/1701.03757.

TRAN, D., KUCUKELBIR, A., DIENG, A. B., RUDOLPH, M., LIANG, D. AND BLEI, D. M. 2016. Edward: A library for probabilistic modeling, inference, and criticism. arXiv preprint arXiv: 1610.09787.

TUCKEY, D., RUSSO, A. AND BRODA, K. 2021. PASOCS: A parallel approximate solver for probabilistic logic programs under the credal semantics, arXiv, abs/2105.10908

VAN DE MEENT, J.-W., PAIGE, B., YANG, H. AND WOOD, F.2021. An introduction to probabilistic programming.

VAN DEN BROECK, G., THON, I., VAN OTTERLO, M. AND DE RAEDT, L. 2010. DTProbLog: A decision-theoretic probabilistic Prolog, 1217–1222.

VIRTANEN, P., GOMMERS, R., OLIPHANT, T. E., HABERLAND, M., REDDY, T., COURNPEAU, D., BUROVSKI, E., PETERSON, P., WECKESSER, W., BRIGHT, J., VAN DER WALT, S. J., BRETT, M., WILSON, J., MILLMAN, K. J., MAYOROV, N., NELSON, A. R. J., JONES, E., KERN, R., LARSON, E., CAREY, C. J., POLAT, İ., FENG, Y., MOORE, E. W., VANDERPLAS, J., LAXALDE, D., PERKTOLD, J., CIMRMAN, R., HENRIKSEN, I., QUINTERO, E. A., HARRIS, C. R., ARCHIBALD, A. M., RIBERIO, A. H., PEDREGOSA, F., VAN MULBERGT, P. AND SciPy 1.0 Contributors 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* 17, 3, 261–272.