A Hybrid Approach to Inference in Probabilistic Non-Monotonic Logic Programming

Matthias Nickles and Alessandra Mileo

Insight Centre for Data Analytics
National University of Ireland, Galway
{matthias.nickles,alessandra.mileo}@deri.org

Abstract. We present a probabilistic inductive logic programming framework which integrates non-monotonic reasoning, probabilistic inference and parameter learning. In contrast to traditional approaches to probabilistic Answer Set Programming (ASP), our framework imposes only comparatively little restrictions on probabilistic logic programs - in particular, it allows for ASP as well as FOL syntax, and for precise as well as imprecise (interval valued) probabilities. User-configurable sampling and inference algorithms, which can be combined in a pipeline-like fashion, provide for general as well as specialized, more scalable approaches to uncertainty reasoning, allowing for adaptability with regard to different reasoning and learning tasks.

1 Introduction

With this paper, we present the probabilistic logic framework PrASP. PrASP is both a probabilistic logic programming language and a software system for probabilistic inference and inductive weight learning based on *Answer Set Programming* (ASP). Compared to previous works on this framework [11, 10], we introduce another inference algorithm and additional evaluation results.
Reasoning in the presence of uncertainty and relational structures such as social networks or Linked Data is an important aspect of knowledge discovery and representation for the Web, the Internet Of Things, and other heterogeneous and complex domains. Probabilistic logic programing, and the ability to learn probabilistic logic programs from data, can provide an attractive approach to uncertainty reasoning and statistical relational learning, since it combines the deduction power and declarative nature of logic programming with probabilistic inference abilities traditionally known from graphical models, such as Bayesian and Markov networks. We build upon existing approaches in the area of probabilistic (inductive) logic programming in order to provide a new ASP-based probabilistic logic programming language and inference tool which combines the benefits of non-monotonic reasoning using state-of-the-art ASP solvers with probabilistic inference and machine learning. The main enhancement provided by PrASP over (non-probabilistic) ASP as well as existing probabilistic approaches to ASP is the possibility to annotate any formula with point or interval (i.e., imprecise) probabilities (including formulas in full FOL syntax, albeit over finite

domains of discourse only), while providing a hybrid set of inference approaches: in addition to general inference algorithms, this includes specialized, more scalable inference algorithms for cases where certain optional assumptions hold (in particular mutual independence of probabilistic events). As we will show later, it can even make sense to combine different such algorithms (which can each on its own obtain valid results if its specific prerequisites are fulfilled).
The remainder of this paper is organized as follows: the next section presents related work. Section 3 describes the syntax and semantics of the formal framework. Section 4 describes approximate inference algorithms, and Section 5 provided initial evaluation results. Section 6 concludes.

2 Related Work

Approaches related to PrASP include [18, 8, 3–6, 14, 13, 15] which support probabilistic inference based on monotonic reasoning and [9, 1, 17, 2] which are based on non-monotonic logic programming. Like P-log [1], our approach computes probability distributions over answer sets (that is, possible worlds are identified with answer sets). However, P-log as well as [17] do not allow for annotating arbitrary formulas (including FOL formulas) with probabilities. [2] allows to associate probabilities with abducibles (only) and to learn both rules and probabilistic weights from given data (in form of literals). Again, PrASP does not impose such restrictions on probabilistic annotations or example data. On the other hand, PrASP cannot make use of abduction for learning. Various less closely related approaches to probabilistic reasoning exist (either not based on logic programming at all, or not in the realm of non-monotonic logic programming): Stochastic Logic Programs (SLP) [8] are an influential approach where sets of rules in form of range-restricted clauses can be labeled with probabilities. Parameter learning for SLPs is approached in [3] using the EM-algorithm. Approaches which combine concepts from Bayesian network theory with relational modeling and learning are, e.g., [4–6]. Probabilistic Relational Models (PRM) [4] can be seen as relational counterparts to Bayesian networks. In contrast to these, our approach does not directly relate to graphical models such as Bayesian or Markov Networks but works on arbitrary possible worlds which are generated by ASP solvers in form of stable models (answer sets). ProbLog [14] allows for probabilistic facts, annotated disjunctions and definite clauses, and approaches to probabilistic rule and parameter learning (from interpretations) also exist for ProbLog. ProbLog builds upon the Distribution Semantics approach introduced for PRISM [18], which is also used by other influential approaches, such as Independent Choice Logic (ICL) [13]. Another important approach outside the area of ASP are Markov Logic Networks (MLN) [15]. A Markov Logic Network consists of first-order formulas annotated with weights (which are, in contrast to PrASP, not in general probabilities). MLNs are used as templates for the construction of Markov networks. The (ground) Markov network generated from the MLN then determines a probability distribution over possible worlds, with inference performed using weighted SAT solving (which is related to but different

from ASP). MLNs are syntactically roughly similar to the logic programs in our framework (where weighted formulas can also be seen as soft or hard constraints for possible worlds).

3 Syntax and Semantics

In this section, we briefly describe the formal language and its semantics. Compared to [10], the syntax of PrASP programs has been extended (in particular by allowing interval and non-ground weights) and a variety of approximate inference algorithms have been added (see next section) to the default inference approach which is described below and which still underlies the formal semantics of PrASP programs.

PrASP is a Nilsson-style [12] probabilistic logic language. Let Φ be a set of function, predicate and object symbols and $\mathcal{L}(\Phi)$ a first-order language over Φ with the usual connectives (including both strong negation "-" and default negation "not") and first-order quantifiers. It can be assumed that this language covers both ASP and FOL syntax (ASP "specialties" such as choice constructs can be seen as syntactic sugar which we omit here in order to keep things simple). A PrASP program (background knowledge) is a non-empty finite set $\Lambda = \{[l_i; u_i]f_i\} \cup \{[l_i; u_i|c_i]f_i\} \cup \{indep(\{f_1^i, ..., f_n^i\})\}$ of annotated formulas (each concluded by a dot) and optional independence constraints (PrASP does not require an independence assumption but makes optionally use of declared or automatically discovered independence). $[l; u]f$ asserts that the imprecise probability of f is within interval $[l, u]$ (i.e., $l \leq Pr(f) \leq u$) whereas $[l; u|c]f$ states that the probability of f conditioned on formula c is within interval $[l, u]$ ($l \leq Pr(f|c) \leq u$).

Formulas can be non-ground (including existentially or universally quantified variables in FOL formulas). For the purpose of this paper, weights need to be ground (real numbers), however, the prototype implementation also allows for certain non-ground weights. An independence constraint $indep(\{f_1^i, ..., f_n^i\})$ specifies that the set of formulas $\{f_1^i, ..., f_n^i\}$ is mutually independent in the probabilistic sense (independence can also be discovered by PrASP by analyzing the background knowledge, but this is computationally more costly).

If the weight of a formula is omitted, $[1; 1]$ is assumed. Point probability weights $[p]$ are translated into weights of the form $[p; p]$ (analogously for conditional probabilities). Weighted formulas can intuitively be seen as constraints which specify which possible worlds (in the form of answer sets) are indeed possible, and with which probability. $w(f)$ denotes the weight of formula f. The f_i and c_i are formulas either in FOL syntax and supported by means of a transformation into ASP syntax described in [7]) or plain AnsProlog syntax, e.g., `[0.5] win :- coin(heads).` Informally, every FOL formula or program with FOL formulas results in a set of ASP formulas. The precise AnsProlog syntax depends on the external ASP grounder being employed by PrASP - in principle, any grounder could be used. The current prototype implementation has been tested with Gringo/Clingo 3 and 4 (`http://potassco.sourceforge.net`).

The semantics of PrASP is defined in terms of probability distributions over possible worlds which are identified with answer sets (models) - an assumption inspired by P-Log [1]. Let $M = (D, \Theta, \pi, \mu)$ be a probability structure where D is a finite discrete domain of objects, Θ is a non-empty set of possible worlds, π is a function which assigns to the symbols in Φ predicates, functions and objects over/from D, and $\mu = (\mu^l, \mu^u)$ is a discrete probability function over Θ, a PrASP program and a query formula, as defined further below.
Each possible world is a Herbrand interpretation over Φ. Since we will use answer sets (i.e., stable models of a (disjunctive) answer set program) as possible worlds, defining $\Gamma(a)$ to be the set of all answer sets of answer set program a will become handy.

We define a (non-probabilistic) satisfaction relation of possible worlds and unannotated programs as follows: let Λ^- be is an unannotated program and lp a transformation which transforms such a program (which might contain formulas in first-order logic syntax in addition to formulas in ASP syntax) into a disjunctive program. The details of this transformation are outside the scope of this paper and can be found in [7].

Then $(M, \theta) \models_\Theta \Lambda^-$ *iff* $\theta \in \Gamma(lp(\Lambda^-))$ and $\theta \in \Theta$. For a disjunctive program ψ, we define $(M, \theta) \models_\Theta \psi$ *iff* $\theta \in \Gamma(\psi)$ and $\theta \in \Theta$.

To do groundwork for the computation of a probability distribution over possible worlds Θ from a given PrASP program, we define a (non-probabilistic) satisfaction relation of possible worlds and unannotated formulas:
Let ϕ be a PrASP formula (without weight) and θ be a possible world. Furthermore, let $(M, \theta) \models_\Lambda \phi$ *iff* $(M, \theta) \models_\Theta \rho(\Lambda) \cup lp(\phi)$ *and* $\Theta = \Gamma(\rho(\Lambda))$ (we say formula ϕ is *true in possible world* θ). Sometimes we will just write $\theta \models_\Lambda \phi$ if M is given by the context. We abbreviate $(M, \theta) \models_\Lambda \phi$ as $\theta \models_\Lambda \phi$. At this, the *spanning program* $\rho(\Lambda)$ of PrASP program Λ is a non-probabilistic disjunctive program (without independence constraints) generated by removing all weights and transforming each formerly weighted formula f or $\neg f$ into a disjunction $f | \neg\, f$, where $\neg$ stands for default negation. Informally, the spanning program represents the uncertain but unweighted beliefs of the knowledge engineer or agent. With $\Gamma(a)$ as defined above, the set of possible worlds deemed possible according to existing belief $\rho(\Lambda)$ is denoted as $\Gamma(\rho(\Lambda))$.
We define the *minimizing* parameterized probability distribution $\mu^l(\Lambda, \Theta, q)$ over a set $\Theta = \{\theta_1, ..., \theta_m\} = \Gamma(\rho(\Lambda))$ of answer sets (possible worlds), a PrASP program $\Lambda = \{([p_i] f_i, i = 1..n)\} \cup \{([p_i | c_i] f_i^c)\} \cup \{indep(\{f_1^i, ..., f_k^i\})\}$ and a query formula q as $\{\theta_i \mapsto Pr(\theta_i) : \theta_i \in \Theta\}$ where $(Pr(\theta_1), ..., Pr(\theta_m))$ is any solution of the following system of inequalities (*constraints*) such that 1) $Pr^l(q) = \sum_{\theta_i \in \Theta : \theta_i \models_\Lambda q} Pr(\theta_i)$ is minimized and 2) the distribution has maximum entropy [19] among any other solutions which minimize the said sum. Analogously, μ^u denotes a maximum entropy probability distribution so that the $Pr(\theta_1), ..., Pr(\theta_m)$ maximize $Pr^u(q) = \sum_{\theta_i \in \Theta : \theta_i \models_\Lambda q} Pr(\theta_i)$.

$$l(f_1) \leq \sum_{\theta_i \in \Theta : \theta_i \models_\Lambda f_1} Pr(\theta_i) \leq u(f_1) \quad \cdots \quad l(f_n) \leq \sum_{\theta_i \in \Theta : \theta_i \models_\Lambda f_n} Pr(\theta_i) \leq u(f_n) \tag{1}$$

$$\sum_{\theta_i \in \Theta} \theta_i = 1 \tag{2}$$

$$\forall \theta_i \in \Theta : 0 \leq Pr(\theta_i) \leq 1 \tag{3}$$

At this, $l(f_i)$ and $u(f_i)$ denote the lower and upper endpoints of the probability interval (imprecise probability) of unconditional formula f_i (analogous for interval endpoints $l(f_i^c|c_i)$ and $u(f_i^c|c_i)$ of conditional probabilities).
In addition, any *indep*-declaration $indep(F^i)$ in the program induces for every subset $\{f_1^i, ..., f_r^i\} \subseteq F^i$, $r > 1$ constraints of the following form:
$\prod_{f_{k=1..r}^i} l(f_k^i) \leq \sum_{\theta_j \in \Theta : \theta_j \vDash_\Lambda \bigwedge f_{k=1..r}^i} Pr(\theta_j) \leq \prod_{f_{k=\{1..r\}}^i} u(f_k^i)$. In the case of point (i.e., precise) probabilities, these encode $Pr(\bigwedge_{k=1..r} f_k^i) = \prod_{k=1..r} Pr(f_k^i)$.
Furthermore, any conditional probability formula $[p_i|c_i]f_i^c)$ in the program induces constraints for ensuring $l(f_i^c|c_i) \leq Pr(f_i^c|c_i) \leq u(f_i^c|c_i)$
(with $p_i = [l(f_i^c|c_i); u(f_i^c|c_i)]$), namely
$\sum_{\theta_j \in \Theta} Pr(\theta_j)\nu(\theta_j, f_i^c \wedge c_i) + \sum_{\theta_j \in \Theta} -l(f_i^c|c_i)Pr(\theta_j)\nu(\theta_j, c_i) > 0$
$\sum_{\theta_j \in \Theta} Pr(\theta_j)\nu(\theta_j, f_i^c \wedge c_i) + \sum_{\theta_j \in \Theta} -u(f_i^c|c_i)Pr(\theta_j)\nu(\theta_j, c_i) < 0$
At this, we define $\nu(\theta, f) = \begin{cases} 1, & \text{if } \theta \vDash_\Lambda f \\ 0, & \text{otherwise} \end{cases}$
For small systems, PrASP can compute minimizing and maximizing probability distributions directly using the inequalities above with linear programming, and a maximum entropy solution amongst a number of candidate distributions (solutions of an underdetermined system) can be discovered using gradient descent. However, to make distribution finding tractable, we need to use different algorithms, as described in the next section. That is, the inequalities system above serves mainly as a means to define the semantics of PrASP formulas.
Finally, marginal inference results are obtained as follows: the result of a query of form `[?] q` is defined as the interval $[Pr^l(q), Pr^u(q)]$ and the result of conditional queries of form `[?|c] f` (which stands for $Pr(f|c)$, where c is some evidence) is computed using $Pr(f \wedge c)/Pr(c)$. An example PrASP program:

```
coin(1..10).
[0.4;0.6] coin_out(1,heads).
[[0.5]] coin_out(N,heads) :- coin(N), N != 1.
1{coin_out(N,heads), coin_out(N,tails)}1 :- coin(N).
n_win :- coin_out(N,tails), coin(N).
win :- not n_win.
[0.8|win] happy.
:- happy, not win.
```

The line starting with `[[0.5]]...` is syntactic sugar for a set of weighted rules where variable `N` is instantiated with all its possible values (i.e.,
`[0.5] coin_out(2,heads) :- coin(2), 2 != 1` and
`[0.5] coin_out(3,heads) :- coin(3), 3 != 1`). It would also be possible to use `[0.5]` as annotation of this rule, in which case the weight 0.5 would specify the probability of the entire non-ground formula instead.
`1{coin_out(N,heads), coin_out(N,tails)}1` (Gringo AnsProlog syntax) denotes that a coin comes up with either heads or tails but not both.

Our system accepts query formulas in format `[?] a`, which asks PrASP for the marginal probability of `a` and `[?|b] a` which computes the conditional probability $Pr(a|b)$. E.g., query `[?|coin_out(2,tails)] happy` results in [0;0].

4 Sampling and Inference Algorithms

PrASP (as a software system) contains a variety of exact and approximate inference algorithms which can be partially combined in a hybrid fashion. Using command line options, the user selects a *pipeline* of alternative pruning (simplification), sampling and inference steps (depending on the nature and complexity of the respective problem). E.g., the user might chose to sample possible worlds from a near-uniform distribution and to pass on the resulting models to a simulated annealing algorithm which computes a probability distribution over the sampled possible worlds. Finally, this distribution is used to compute the conditional or marginal probabilities of the query formulas. The inference algorithms available in the current prototype (version 0.7) of PrASP are:

Linear programming Direct solution for the linear inequalities system described before. Precise and very fast for very small systems, intractable otherwise.

Various answer set sampling algorithms for so-called *initial sampling* These can in some cases be used directly for inference, by computing a distribution which complies with the constraints (linear system) described before. An exemplary such algorithm is Algorithm 1. Alternatively, they can be followed by another inference algorithm (simulated annealing or iterative refinement, see below) which corrects the initial distribution computed by initial sampling.

Parallel simulated annealing This approach (Algorithm 2) performs simulated annealing for inference problems where no assumptions can be made about independence or other properties of the program (except consistency). It can be used either stand-alone or in a hybrid combination with an initial sampling stage (e.g., Algorithm 1).

Iterative refinement An adaptation of the inference algorithm described in [16] with guaranteed minimal Kullback−Leibler divergence to the uniform distribution (i.e., maximum entropy).

Direct counting Weights are transformed into unweighted formulas and queries are then solved by mere counting of models (see [10] for details).

Most of our algorithms rely heavily on near-uniform sampling, either using randomization provided by the respective external ASP solver (fast but typically rather low quality, i.e., weakly uniform) or using so-called XOR-constraints as described in [10] (which provides higher sampling quality at expense of speed).
From PrASP's inference algorithms, we describe one of the initial sampling algorithms (Algorithm 1) and parallel simulated annealing (Algorithm 2).
An interesting property of the first algorithm is its ability to provide a suitable distribution over possible worlds directly if all weighted formulas in the PrASP program are mutually independent (analogously to the independence assumption typically made by distribution semantics-based approaches). Algo. 2 can be

used stand-alone or subsequently to Algo. 1: in that case, the probability distribution computer by initial sampling (with replacement) is used as the initial distribution which is then refined by simulated annealing until all constraints (given probabilities) are fulfilled. The benefit of this pipelined approach to inference is that the user (knowledge engineer) doesn't need to know about event independence - if the uncertain formulas in the program are independent, initial sampling already provides a valid distribution and the subsequent simulated annealing stage almost immediately completes. Otherwise, simulated annealing "repairs" the insufficient distribution computed by the initial sampling stage.
Concretely, Algo. 1 samples answer sets and computes a probability distribution over these models which reflects the weights provided in the PrASP program, provided that all uncertain formulas in the program describe a mutually independent set of events. Other user-provided constraints (such as conditional probabilities in the PrASP program) are ignored here. Also, Algo. 1 does not guarantee that the solution has maximum entropy.

Algorithm 1 Sampling from models of spanning program (point probabilities only)

Require: max number of samples n, set of *uncertain* formulas $uf = \{[w(uf_i)]uf_i \text{ with } 0 < w(uf_i) < 1\}$, set of *certain* formulas $cf = \{cf_i : w(uf_i) = 1\}$ (i.e., with probability 1)

$i \leftarrow 1$
for $i \leq |uf|$ **do**
  $r^i \leftarrow$ random element of $Sym(\{1, ..., n\})$ (permutations of $\{1, ..., n\}$)
  $i \leftarrow i + 1$
end for
$m \leftarrow \emptyset$, $j \leftarrow 1$
parfor $j \in \{1, ..., n\}$ **do**
  $p \leftarrow \emptyset$, $k \leftarrow 1$
  for $k \leq |uf|$ **do**
    if $r^k_j \leq n \cdot w(uf_k)$ **then** $p \leftarrow p \cup \{uf_k\}$ **else** $p \leftarrow p \cup \{\neg uf_k\}$ **endif**
    $k \leftarrow k + 1$
  end for
  $s \leftarrow$ model sampled uniformly from models of program $cf \cup p$ ($\emptyset$ if UNSAT)
  $m \leftarrow m \uplus \{s\}$
end parfor

Ensure: Multiset m contains samples from all answer sets of spanning program such that

$\forall uf_i : w(uf_i) \approx \frac{|\{s \in m : s \models uf_i\}|}{|m|}$ *iff* set uf mutually independent.

Algorithm 2 presents the approach PrASP uses for approximate inference using a parallel form of simulated annealing (which does not require event independence). The initial list of samples *initSamples* are computed according to an initial sampling approach such as Algo. 1.

Algorithm 2 Inference by parallel simulated annealing Part 1.
We show only the basic variant for non-conditional formulas with point weights. The extension for conditional probabilities and interval probabilities (imprecise probabilities) is straightforward.

Require: *maxTime*, *maxEnergy*, *initTemp*, *initSamples* (from, e.g., Algo. 1). *initSamples* is a multiset which encodes a probability distribution via frequencies of models), *frozen*, *degreeOfParallelization*, α, F (set of weighted formulas), Λ (PrASP program)

$s \leftarrow initSamples$, $k \leftarrow 0$, $temp \leftarrow initTemp$
$e \leftarrow \text{ENERGY}(s)$
while $k \leq maxTime \wedge temp \geq frozen$ **do**
  parfor $i \leftarrow 1, degreeOfParallelization$ **do**
    $s''_i \leftarrow s'' \uplus \text{SAMPLESTEP}(samplingMethod)$
  end parfor
  $s' \leftarrow \text{argmin}_{s''}(\text{ENERGY}(s''_1), ..., \text{ENERGY}(s''_n))$
  $e' \leftarrow \text{ENERGY}(s')$
  if $e' < e \vee random_0^1 < e^{-(e'-e)/temp)}$ **then**
    $s \leftarrow s'$
    $e \leftarrow e'$
  end if
  $temp \leftarrow temp \cdot \alpha$
  $k \leftarrow k + 1$
end while

Ensure: Multiset $s = (pw, frq) = \mu_{approx}(\Lambda)$ approximates the probability distribution $\mu(\Lambda) = Pr(\Gamma(\rho(\Lambda)))$ over the set $pw = \{pw_i\} = \Gamma(\rho(\Lambda))$ of possible worlds by $\{Pr(pw_i) \approx \frac{frq(pw)}{|s|}\}$.

function ENERGY(s)
  parfor $f_i \in |F|$ **do**
    $freq_{f_i} \leftarrow \frac{|\{\{s' \in s : s' \models_\Lambda f_i\}\}|}{|s|}$
  end parfor
  return $\sqrt{\sum_{f_i \in F} (freq_{f_i} - weight_{f_i})^2}$
end function
▷ (Continued in Part 2 below)

In addition to the actual inference algorithms, it is often beneficial to let PrASP remove (prune) all parts of the spanning program which cannot influence the probabilities of the query formulas. The approach to this is straightforward (program dependency analysis) and therefore omitted here. We will show in the evaluation section how such simplification affects inference performance.

Algorithm 2 Inference by parallel simulated annealing Part 2.

22: **function** STEPSAMPLE(*samplingMethod*)
▷ The framework provides various configurable methods for the simulated annealing sampling step, of which we show here only one.
23: $F' \leftarrow \emptyset$
24: **for** $f_i \in |F|$ **do**
25: **if** $random_0^1 < weight_{f_i}$ **then**
26: $F' \leftarrow F' \cup \{f_i\}$
27: **else**
28: $F' \leftarrow F' \cup \{\neg f_i\}$
29: **end if**
30: **end for**
return $answerSets(F')$ (might be $\emptyset$)
31: **end function**

While for space-related reasons this paper covers deductive inference only, PrASP also supports induction (learning of weights of hypotheses from example data). Please refer to [10] for details.

5 Experiments

The main goal of PrASP is not to outperform existing approaches in terms of speed but to provide a flexible, scalable and highly configurable framework which puts as few restrictions as possible on what users can express in terms of (non-monotonic) certain and uncertain beliefs while being competitive with more specialized inference approaches if the respective conditions (like event independence) are met.
For our first experiment, we model a coin game (a slightly simplified variant of the example code shown before): a number of coins are tossed and the game is won if a certain subset of all coins comes up with "heads". The inference task is the approximation of the winning probability. In addition, another random subset of coins are magically dependent from each other and one of the coins is biased (probability of "heads" is 0.6). Despite its simplicity, this scenario shows how inference copes with independent as well as dependent uncertain facts, and how effective the pruning approach of the respective framework works (since winning depends only on a subset of coins). Also, inference complexity clearly scales with the number of coins. In PrASP syntax, such a partially randomly generated program looks, e.g., as follows (adaptation to MLN or ProbLog syntax is straightforward):

```
coin(1..8).
[0.6] coin_out(1,heads).
[[0.5]] coin_out(N,heads) :- coin(N), N != 1.
1{coin_out(N,heads), coin_out(N,tails)}1 :- coin(N).
win :- 2{coin_out(3,heads),coin_out(4,heads)}2.
coin_out(4,heads) :- coin_out(6,heads).
```

The inference algorithm used is initial sampling (Algo. 1) followed by simulated annealing (Algo. 2). Using Algo. 1, we computed 100 random models (number of samples n), which is sufficient to obtain a precision of +/-0.01 for the query probabilities. The winning subset of coins and the subset of mutually dependent coins (from which a rule of the form

```
coin_out(a,heads) :- coin_out(b,heads), coin_out(c,heads), ...
```

is generated) is each a random set with 25% of the size of the respective full set of coins. "PrASP 0.7.2 simp" in Fig. 1 stands for results obtained with pruning (i.e., parts of the program on which the query result cannot depend have been automatically removed). We also report the results obtained solely using (Algorithm 2) ("noinit" in Fig. 1), in order to see whether the initial sampling stage provides any benefits here. Simulated annealing parameters have been $maxEnergy = 0.15$, $initTemp = 5$, $frozen = 10^{-150}$, $\alpha = 0.85$.

We compared the performance (duration in dependency of the number of coins (x-axis), minimum number of 18 coins) of the current prototype of PrASP with that of Tuffy 0.3 (http://i.stanford.edu/hazy/hazy/tuffy/), a recent implementation of Markov Logic Networks which uses a database system in order to increase scalability, and ProbLog2 2.1 (https://dtai.cs.kuleuven.be/problog/) (despite random dependencies). Times are in milliseconds, obtained using an i7 4-cores processor with 3.4GHz over five trials.

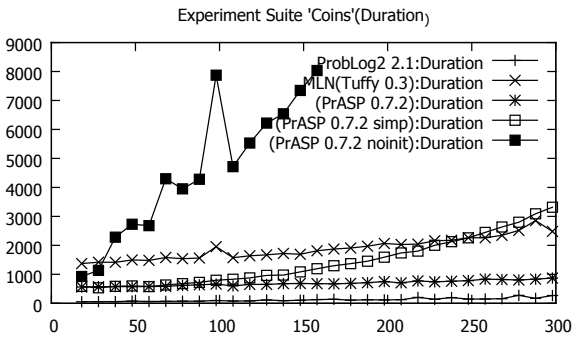

Fig. 1. Biased coins game

ProbLog2 and Tuffy scale very well here, with some additional time required by Tuffy probably due to the overhead introduced by external database operations. With PrASP, we observe that priming simulated annealing with an initial sampling step (which would give inaccurate results if used standalone) improves performance massively, whereas pruning appears to suffer from the additional overhead introduced by the required dependency analysis of the logic program. Our hypothesis regarding this behavior is that the initial distribution over possible worlds is, albeit not perfect, quite close to the accurate distribution so that the subsequent simulated annealing task takes off a lot faster compared to starting from scratch (i.e., from the uniform distribution).

The next experiment shows how PrASP copes with a more realistic benchmark task - a form of the well-known friends-and-smokers problem [15] - which can be tractably approached using Algorithm 1 alone since the independence assumption is met (which also makes it suitable for ProbLog). On the other hand, the rules are more complex. In this benchmark scenario, a randomly chosen number of persons are friends, a randomly chosen subset of all people

smoke, there is a certain probability for being stressed (`[[0.3]] stress(X)`), it is assumed that stress leads to smoking (`smokes(X) :- stress(X)`), and that some friends influence each other with a certain probability (`[[0.2]] influences(X,Y)`), in particular with regard to their smoking behavior `smokes(X) :- friend(X,Y), influences(Y,X), smokes(Y)`. With a certain probability, smoking leads to asthma (`[[0.4]] h(X). asthma(X) :- smokes(X), h(X)`). The query comprises of `[?] asthma(X)` for each person X.

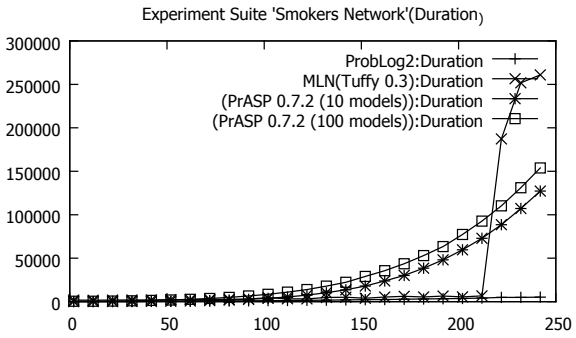

Fig. 2. Smokers social network

The results (Fig. 2) have been averaged over five trials. Again, ProbLog2 scores best in this scenario. PrASP, using Algorithm 1 (since all uncertain facts in this scenario are mutually independent), does quite well for most of episodes but looses on ProbLog2. Tuffy does very well below 212 persons, then performance massively breaks in for unknown reasons (possibly due to some internal cache overflow). For technical reasons, we couldn't get the smokers-scenario working with the publicly available current implementation of P-Log (we received segmentation faults which couldn't be resolved), but experiments with examples coming with this software seem to indicate that this approach also scales fine.

6 Conclusion

We have presented a new software framework for uncertainty reasoning and parameter estimation based on Answer Set Programming. In contrast to most other approaches to probabilistic logic programming, the philosophy of PrASP is to provide a very expressive formal language (ASP or full FOL syntax over finite domains for formulas annotated with precise as well as imprecise probabilities) on the one hand and a variety of inference algorithms which are able to take advantage of certain problem domains which facilitate "fast track" reasoning and learning (in particular inference in the presence of formula independence) on the other. We see the main benefit of our framework, besides its support for non-monotonic reasoning, thus in its semantically rich and configurable uncertainty reasoning approach which allows to combine various sampling and inference approaches in a pipeline-like fashion. Ongoing work focuses on additional experiments and the integration of further inference algorithms, and the direct integration of an ASP solver into PrASP, in order to avoid expensive calls of external reasoning tools. Another area of ongoing work is the support for so-called annotated disjunctions [20]. **Sponsored by SFI grant n. SFI/12/RC/2289.**

References

1. Baral, C., Gelfond, M., Rushton, N.: Probabilistic reasoning with answer sets. Theory Pract. Log. Program. 9(1), 57–144 (2009)
2. Corapi, D., Sykes, D., Inoue, K., Russo, A.: Probabilistic rule learning in nonmonotonic domains. In: Procs. 12th international conference on Computational logic in multi-agent systems. pp. 243–258. CLIMA'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2044543.2044565
3. Cussens, J.: Parameter estimation in stochastic logic programs. In: Mach. Learn. p. 2001 (2000)
4. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: In IJCAI. pp. 1300–1309. Springer-Verlag (1999)
5. Kersting, K., Raedt, L.D.: Bayesian logic programs. In: Proceedings of the 10th International Conference on Inductive Logic Programming (2000)
6. Laskey, K.B., Costa, P.C.: Of klingons and starships: Bayesian logic for the 23rd century. In: Procs. of the 21st Conf. on Uncertainty in Artificial Intelligence (2005)
7. Lee, J., Palla, R.: System f2lp - computing answer sets of first-order formulas. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR. Lecture Notes in Computer Science, vol. 5753, pp. 515–521. Springer (2009)
8. Muggleton, S.: Learning stochastic logic programs. Electron. Trans. Artif. Intell. 4(B), 141–153 (2000)
9. Ng, R.T., Subrahmanian, V.S.: Stable semantics for probabilistic deductive databases. Inf. Comput. 110(1), 42–83 (1994)
10. Nickles, M., Mileo, A.: Probabilistic inductive logic programming based on answer set programming. In: 15th Int'l Workshop on Non-Monotonic Reasoning (NMR'14) (2014)
11. Nickles, M., Mileo, A.: A system for probabilistic inductive answer set programming. In: 9th International Conference on Scalable Uncertainty Management (SUM'15) (2015, to appear)
12. Nilsson, N.J.: Probabilistic logic. Artificial Intelligence 28(1), 71–87 (1986)
13. Poole, D.: The independent choice logic for modelling multiple agents under uncertainty. Artificial Intelligence 94, 7–56 (1997)
14. Raedt, L.D., Kimmig, A., Toivonen, H.: Problog: A probabilistic prolog and its application in link discovery. In: IJCAI. pp. 2462–2467 (2007)
15. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learning 62(1-2), 107–136 (February 2006), http://dx.doi.org/10.1007/s10994-006-5833-1
16. Rodder, W., Meyer, C.: Coherent knowledge processing at maximum entropy by spirit. In: Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI'96), 1996 (1996)
17. Saad, E., Pontelli, E.: Hybrid probabilistic logic programming with non-monotoic negation. In: In Twenty First International Conference on Logic Programming. Springer Verlag (2005)
18. Sato, T., Kameya, Y.: Prism: a language for symbolic-statistical modeling. In: In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI97. pp. 1330–1335 (1997)
19. Thimm, M., Kern-Isberner, G.: On probabilistic inference in relational conditional logics. Logic Journal of the IGPL 20(5), 872–908 (2012)
20. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) Logic Programming, Lecture Notes in Computer Science, vol. 3132, pp. 431–445. Springer Berlin Heidelberg (2004)