Probabilistic Logic Programming in Action

Arnaud Nguembang Fadja[1] and Fabrizio Riguzzi[2]

[1] Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[2] Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[fabrizio.riguzzi,arnaud.nguembangfadja]@unife.it

Abstract. Probabilistic Programming (PP) has recently emerged as an effective approach for building complex probabilistic models. Until recently PP was mostly focused on functional programming while now Probabilistic Logic Programming (PLP) forms a significant subfield. In this paper we aim at presenting a quick overview of the features of current languages and systems for PLP. We first present the basic semantics for probabilistic logic programs and then consider extensions for dealing with infinite structures and continuous random variables. To show the modeling features of PLP in action, we present several examples: a simple generator of random 2D tile maps, an encoding of Markov Logic Networks, the truel game, the coupon collector problem, the one-dimensional random walk, latent Dirichlet allocation and the Indian GPA problem. These examples show the maturity of PLP.

Keywords: Probabilistic Logic Programming, Probabilistic Logical Inference, Hybrid programs

1 Introduction

Probabilistic Logic Programming (PLP) [11] models domains characterized by complex and uncertain relationships among domain entities by combining probability theory with logic programming. The field started in the early nineties with the seminal work of Dantsin [8], Poole [33] and Sato [49] and is now well established, with a dedicated annual workshop since 2014.

PLP has been applied successfully to many problems such as concept relatedness in biological networks [12], Mendel's genetic inheritance [50], natural language processing [51,44], link prediction in social networks [24], entity resolution [37] and model checking [15].

PLP is a type of Probabilistic Programming (PP) [30], a collection of techniques that have recently emerged as an effective approach for building complex probabilistic models. Until recently PP was mostly focused on functional programming while now PLP forms a significant subfield.

Various approaches have been proposed for representing probabilistic information in Logic Programming [27,10]. The distribution semantics [49] is one of the most used and underlies many languages, such as Independent Choice Logic [32], PRISM [49], Logic Programs with Annotated Disjunctions (LPADs) [56] and ProbLog [12]. While these languages differ syntactically, they have the same expressive power, as there are linear transformations among them [55].

In this paper we aim at giving an overview of the current status of PLP. We start by first presenting the semantics for programs without function symbols and then discussing the extensions of this semantics for programs including function symbols, that may have infinite computation branches, and for programs including continuous random variables, a recent proposal that brought PLP closer to functional PP approaches, where continuous random variables have been a basic feature for some time now.

We also discuss approaches for inference, starting from exact inference by knowledge compilation and going to techniques for dealing with infinite computation branches and continuous random variables. Then we illustrate approximate inference approaches based on Monte Carlo methods that can overcome some of the limit of exact inference both in terms of computation time and allowed language features, permitting inference on a less restricted class of programs.

To show the modeling features of PLP in action, we present several examples. We start from a simple generator of random 2D tile maps for video games and from an approach for encoding Markov Logic Networks, a popular Statistical Relation Artificial Intelligence formalism. In both examples all the variables are discrete and no infinite computation path exists. We then consider three domains with possibly infinite computations: the truel problem from game theory, the coupon collector problem and the one-dimensional random walk. Finally we discuss two examples where some of the variables are continuous: latent Dirichlet allocation and the Indian GPA problem. For each example we provide the code for the `cplint` system [1] together with links to the web application `cplint` on SWISH `http://cplint.ml.unife.it` [39,47] where the examples can be run online, without the need to install the system locally.

PLP has found important applications in the field of Machine Learning and Knowledge Extraction [17]: it is used as the language for representing input data and output models in a variety of Machine Learning systems [40,14,43,35] that have already achieved a significant number of successful applications. The web application `cplint` on SWISH also includes the learning systems EMBLEM [4,3], SLIPCOVER [5] and LEMUR [13]. For lack of space we do not discuss learning in this paper.

2 Syntax and Semantics

We consider first the discrete version of probabilistic logic programming languages. In this version, each logical atom is a Boolean random variable that can assume values true or false. The facts and rules of the program specify the dependences among the truth values of atoms and the main inference task is to compute the probability that a ground query is true, often conditioned on the truth of a conjunction of ground goals, the evidence. All the languages following the distribution semantics allow the specification of alternatives either for facts and/or for clauses. We present here the syntax of LPADs [56] for its generality.

An LPAD is a finite set of annotated disjunctive clauses of the form

$$h_{i1} : \Pi_{i1}; \ldots ; h_{in_i} : \Pi_{in_i} \text{ :- } b_{i1}, \ldots , b_{im_i}.$$

where $b_{i1}, \ldots , b_{im_i}$ are literals, $h_{i1}, \ldots h_{in_i}$ are atoms and $\Pi_{i1}, \ldots , \Pi_{in_i}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. This clause can be interpreted as "if $b_{i1}, \ldots , b_{im_i}$ is true, then h_{i1} is true with probability Π_{i1} or ...or h_{in_i} is true with probability Π_{in_i}." $b_{i1}, \ldots , b_{im_i}$ is the body of the clause and we indicate it with $body(C)$ if the clause is C. If $n_i = 1$ and $\Pi_{i1} = 1$ the clause is non-disjunctive and so not probabilistic. If $\sum_{k=1}^{n_i} \Pi_{ik} < 1$, there is an implicit annotated atom $null : (1 - \sum_{k=1}^{n_i} \Pi_{ik})$ that does not appear in the body of any clauses of the program.

Given an LPAD P, the grounding $ground(P)$ is obtained by replacing variables with all possible logic terms. If P does not contain function symbols, the set of possible terms is equal to the set of all constants appearing in P and is finite so $ground(P)$ is finite as well.

$ground(P)$ is still an LPAD from which, by selecting a head atom for each ground clause, we can obtain a normal logic program, called "world". In the distribution semantics, the choices of head atoms for different clauses are independent, so we can assign a probability to a world by multiplying the probabilities of all the head atoms chosen to form the world. In this way we get a probability distribution over worlds from which we can define a probability distribution over the truth values of a ground atom: the probability of an atom q being true is the sum of the probabilities of the worlds where q is true in the well-founded model [54] of the world.

The well-founded model [54] in general is three valued, so a query that is not true in the model is either undefined or false. However, we consider atoms as Boolean random variables so we do not want to deal with the undefined truth value. How to manage uncertainty by combining nonmonotonic reasoning and probability theory is still an open problem, see [7] for a discussion of the issues. So we require each world to have a two-valued well-founded model and therefore q can only be true or false in a world.

Formally, each grounding of a clause $C_i\theta_j$ corresponds to a random variable X_{ij} with as many values as the number of head atoms of C_i. The random variables X_{ij} are independent of each other.

An *atomic choice* [31] is a triple (C_i, θ_j, k) where $C_i \in P$, θ_j is a substitution that grounds C_i and $k \in \{1, \ldots , n_i\}$ identifies one of the head atoms. In practice $C_i\theta_j$ corresponds to an assignment $X_{ij} = k$. A set of atomic choices κ is *consistent* if only one head is selected for the same ground clause. A consistent set κ of atomic choices is called a *composite choice*. We can assign a probability to κ as the random variables are independent: $P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \Pi_{ik}$.

A *selection* σ is a composite choice that, for each clause $C_i\theta_j$ in $ground(P)$, contains an atomic choice (C_i, θ_j, k). A selection σ identifies a normal logic program l_σ defined as $l_\sigma = \{(h_{ik} \leftarrow body(C_i))\theta_j|(C_i, \theta_j, k) \in \sigma\}$. l_σ is called an *instance*, *possible world* or simply *world* of P. Since selections are composite choices, we can assign a probability to instances: $P(l_\sigma) = P(\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$.

We write $l_\sigma \models q$ to mean that the query q (a ground atom) is true in the well-founded model of the program l_σ. The probability of a query q given a world l_σ can be now defined as $P(q|l_\sigma) = 1$ if $l_\sigma \models q$ and 0 otherwise. Let $P(L_P)$ be the distribution over worlds. The probability of a query q is given by

$$P(q) = \sum_{l_\sigma \in L_P} P(q, l_\sigma) = \sum_{l_\sigma \in L_P} P(q|l_\sigma)P(l_\sigma) = \sum_{l_\sigma \in L_P : l_\sigma \models q} P(l_\sigma) \qquad (1)$$

Example 1 (From [5]). The following LPAD P encodes geological knowledge on the Stromboli Italian island:

$C_1 = eruption : 0.6 \;;\; earthquake : 0.3 :- \; sudden\_energy\_release,$
$\quad fault\_rupture(X).$
$C_2 = sudden\_energy\_release : 0.7.$
$C_3 = fault\_rupture(southwest\_northeast).$
$C_4 = fault\_rupture(east\_west).$

The Stromboli island is located at the intersection of two geological faults, one in the southwest-northeast direction, the other in the east-west direction, and contains a active volcano. This program models the possibility that an eruption or an earthquake occurs at Stromboli. If there is a sudden energy release under the island and there is a fault rupture, then there can be an eruption of the volcano on the island with probability 0.6 or an earthquake in the area with probability 0.3 or no event with probability 0.1. The energy release occurs with probability 0.7 while we are sure that ruptures occur in both faults.

Clause C_1 has two groundings, $C_1\theta_1$ with $\theta_1 = \{X/southwest\_northeast\}$ and $C_1\theta_2$ with $\theta_2 = \{X/east\_west\}$, so there are two random variables X_{11} and X_{12}. Clause C_2 has only one grounding $C_2\emptyset$ instead, so there is one random variable X_{21}. X_{11} and X_{12} can take three values since C_1 has three head atoms; similarly X_{21} can take two values since C_2 has two head atoms. P has 18 instances, the query *eruption* is true in 5 of them and its probability is
$P(eruption) = 0.6\cdot0.6\cdot0.7+0.6\cdot0.3\cdot0.7+0.6\cdot0.1\cdot0.7+0.3\cdot0.6\cdot0.7+0.1\cdot0.6\cdot0.7 = 0.588$.

This semantics can be given also a sampling interpretation: the probability of a query q is the fraction of worlds, sampled from the distribution over worlds, where q is true. To sample from the distribution over worlds, you simply randomly select a head atom for each ground clause according to the probabilistic annotations. Note that you don't even need to sample a complete world: if the samples you have taken ensure the truth value of q is determined, you don't need to sample more clauses, as they don't influence q.

To compute the conditional probability $P(q|e)$ of a query q given evidence e, you can use the definition of conditional probability, $P(q|e) = P(q, e)/P(e)$, and compute first the probability of q, e (the sum of probabilities of worlds where both q and e are true) and the probability of e and then divide the two.

If the program P contains function symbols, a more complex definition of the semantics is necessary. In fact $ground(P)$ is infinite and a world would be obtained by making an infinite number of choices so its probability would be 0, as it is a product of infinite numbers all bounded away from 1 from below. In this case we have to work with sets of worlds and use Kolmogorov's definition of probability space. It turns out that the probability of the query is the sum of a convergent series [38].

Up to now we have considered only discrete random variables and discrete probability distributions. How can we consider continuous random variables and probability density functions, for example real variables following a Gaussian distribution? `cplint` [1] and Distributional Clauses (DC) [28] allow the description of continuous random variables in so called *hybrid programs*.

`cplint` allows the specification of density functions over arguments of atoms in the head of rules. For example, in

```
g(X,Y): gaussian(Y,0,1):- object(X).
```

`X` takes terms while `Y` takes real numbers as values. The clause states that, for each `X` such that `object(X)` is true, the values of `Y` such that `g(X,Y)` is true, follow a Gaussian distribution with mean 0 and variance 1. You can think of an atom such as `g(a,Y)` as an encoding of a continuous random variable associated to term `g(a)`. In DC you can express the same density as

```
g(X)~gaussian(0,1):= object(X).
```

where `:=` indicates implication and continuous random variables are represented as terms that denote a value from a continuous domain. It is possible to translate DC into programs for `cplint` and in fact `cplint` allows also the DC syntax, automatically translating DC into its own syntax.

A semantics for hybrid programs was given independently in [16,28] and [20]. In [28] the semantics of Hybrid Probabilistic Logic Programs (HPLP) is defined by means of a stochastic generalization STp of the Tp operator that applies the sampling interpretation of the distribution semantics to continuous variables: STp is applied to interpretations that contain ground atoms (as in standard logic programming) and terms of the form $t = v$ where t is a term indicating a continuous random variable and v is a real number. If the body of a clause is true in an interpretation I, $STp(I)$ will contain a sample from the head.

The authors of [20] define a probability space for N continuous random variables by considering the Borel σ-algebra over $\mathbb{R}^N$ and fixing a Lebesgue measure on this set as the probability measure. The probability space is lifted to cover the entire program using the least model semantics of constraint logic programs.

If an atom encodes a continuous random variable (such as `g(X,Y)` above), asking for the probability that a ground instantiation, such as `g(a,0.3)`, is true is not meaningful, as the probability that a continuous random variables takes a specific value is always 0. In this case you want to compute the probability that the random variable falls in an interval or you want to know its density, possibly after having observed some evidence. If the evidence is on an atom defining another continuous random variable, the definition of conditional probability cannot be applied, as the probability of the evidence would be 0 and so the fraction would be undefined. This problem is tackled in [28] by providing a definition using limits.

3 Inference

Computing all the worlds is impractical because their number is exponential in the number of ground probabilistic clauses when there are no function symbols and impossible otherwise, because with function symbols the number of worlds is uncountably infinite. Alternative approaches for inference have been considered that can be grouped in exact and approximate ones [1].

For exact inference from discrete program without function symbols a successful approach finds explanations for the query q [12], where an explanation is a set of clause choices that are sufficient for entailing the query. Once all explanations for the query are found, they are encoded as a Boolean formula in DNF and the problem is reduced to that of computing the probability that the formula is true given the probabilities of being true of all the (mutually independent) random variables. This problem is called *disjoint-sum* as it can be solved by finding a DNF where all the disjuncts are mutually exclusive. Its complexity is #P [53] so the problem is highly difficult and intractable in general. In practice, problems of significant size can be tackled using *knowledge compilation* [9], i.e. converting the DNF into a language from which the computation of the probability is polynomial [12,45], such as Binary Decision Diagrams.

Formally, a composite choice κ is an *explanation* for a query q if q is entailed by every instance consistent with κ, where an instance l_σ is consistent with κ iff $\kappa \subseteq \sigma$. Let λ_κ be the set of worlds consistent with κ. In particular, algorithms find a covering set of explanations for the query, where a set of composite choices K is *covering* with respect to q if every program l_σ in which q is entailed is in λ_K, where $\lambda_K = \sum_{\kappa \in K} \lambda_\kappa$. The problem of computing the probability of a query q can thus be reduced to computing the probability of the Boolean function

$$f_q(\mathbf{X}) = \bigvee_{\kappa \in E(q)} \bigwedge_{(C_i,\theta_j,k)\in\kappa} X_{ij} = k \tag{2}$$

where $E(q)$ is a covering set of explanations for q.

Example 2 (Example 1 cont.). The query *eruption* has the covering set of explanations $E(eruption) = \{\kappa_1, \kappa_2\}$ where:

$$\kappa_1 = \{(C_1, \{X/southwest\_northeast\}, 1), (C_2, \{\}, 1)\}$$
$$\kappa_2 = \{(C_1, \{X/east\_west\}, 1), (C_2, \{\}, 1)\}$$

Each atomic choice (C_i, θ_j, k) is represented by the propositional equation $X_{ij} = k$:

$$\begin{array}{ll} (C_1, \{X/southwest\_northeast\}, 1) & \rightarrow X_{11} = 1 \\ (C_1, \{X/east\_west\}, 1) & \rightarrow X_{12} = 1 \\ (C_2, \{\}, 1) & \rightarrow X_{21} = 1 \end{array}$$

The resulting Boolean function $f_{eruption}(\mathbf{X})$ returns 1 if the values of the variables correspond to an explanation for the goal. Equations for a single explanation are conjoined and the conjunctions for the different explanations are disjoined. The set of explanations $E(eruption)$ can thus be encoded with the function:

$$f_{eruption}(\mathbf{X}) = (X_{11} = 1 \wedge X_{21} = 1) \vee (X_{12} = 1 \wedge X_{21} = 1) \tag{3}$$

Examples of systems that perform inference using this approach are ProbLog [23] and PITA [45,46]. Recent approaches for exact inference try to achieve speedups by reasoning at a lifted level [2,41].

When a discrete program contains function symbols, the number of explanations may be infinite and the probability of the query may be the sum of a convergent series. In this case the inference algorithm has to recognize the presence of an infinite number of explanations and identify the terms of the series. In [52,48] the authors extended PRISM by considering programs under the *generative exclusiveness condition*: at any choice point in any execution path of the top-goal, the choice is done according to a value sampled from a PRISM probabilistic switch. The generative exclusiveness condition implies that every disjunction is exclusive and originates from a probabilistic choice made by some switch.

In this case, a *cyclic explanation graph* can be computed that encodes the dependence of atoms on probabilistic switches. From this a system of equations can be obtained defining the probability of ground atoms. The authors of [52,48] show that by first assigning all atoms probability 0 and repeatedly applying the equations to compute updated values results in a process that converges to a solution of the system of equations. For some program, such as those computing the probability of prefixes of strings from Probabilistic Context Free Grammars, the system is linear, so solving it is even simpler. In general, this provides an approach for performing inference when the number of explanations is infinite but under the generative exclusiveness condition.

In [15] the authors present the algorithm PIP (for Probabilistic Inference Plus), that is able to perform inference even when explanations are not necessarily mutually exclusive and the number of explanations is infinite. They require the programs to be *temporally well-formed*, i.e., that one of the arguments of predicates can be interpreted as a time that grows from head to body. In this case the explanations for an atom can be represented succinctly by Definite Clause Grammars (DCGs). Such DCGs are called *explanation generators* and are used to build Factored Explanation Diagrams (FED) that have a structure that closely follows that of Binary Decision Diagrams. FEDs can be used to obtain a system of polynomial equations that is monotonic and thus convergent as in [52,48]. So, even when the system is non linear, a least solution can be computed to within an arbitrary approximation bound by an iterative procedure.

For approximate inference one of the most used approach consists in Monte Carlo sampling, following the sampling interpretation of the semantics given above. Monte Carlo backward reasoning has been implemented in ProbLog [23] and MCINTYRE [36] and found to give good performance in terms of quality of the solutions and of running time. Monte Carlo sampling is attractive for the simplicity of its implementation and because you can improve the estimate as more time is available. Moreover, Monte Carlo can be used also for programs with function symbols, in which goals may have infinite explanations and exact inference may loop. In fact, taking a sample of a query corresponds naturally to an explanation and the probability of a derivation is the same as the probability of the corresponding explanation. The risk is that of incurring in an infinite explanation. But infinite explanations have probability 0 so the probability that the computation goes down such a path and does not terminate is 0 as well.

Monte Carlo inference provides also smart algorithms for computing conditional probabilities: rejection sampling or Metropolis-Hastings Markov Chain Monte Carlo (MCMC). In rejection sampling [57], you first query the evidence and, if the query is successful, query the goal in the same sample, otherwise the sample is discarded.

In Metropolis-Hastings MCMC [26], a Markov chain is built by taking an initial sample and by generating successor samples. The initial sample is built by randomly sampling choices so that the evidence is true. A successor sample is obtained by deleting a fixed number of sampled probabilistic choices. Then the evidence is queried again by sampling starting with the undeleted choices. If the query succeeds, the goal is then also queried by sampling. The goal sample is accepted with a probability of $\min\{1, \frac{N_0}{N_1}\}$ where N_0 is the number of choices sampled in the previous sample and N_1 is the number of choices sampled in the current sample. The number of successes of the query is increased by 1 if the query succeeded in the last accepted sample. The final probability is given by the number of successes over the total number of samples.

When you have evidence on ground atoms that have continuous values as arguments, you can still use Monte Carlo sampling. You cannot use rejection sampling or Metropolis-Hastings, as the probability of the evidence is 0, but you can use likelihood weighting [28] to obtain weighted samples of continuous arguments of a goal. For each sample to be taken, likelihood weighting samples the query and then assigns a weight to the sample on the basis of evidence. The weight is computed by deriving the evidence backward in the same sample of the query starting with a weight of one: each time a choice should be taken or a continuous variable sampled, if the choice/variable has already been taken, the current weight is multiplied by probability of the choice/by the density value of the continuous value.

If likelihood weighting is used to find the posterior density of a continuous random variable, we obtain a set of weighted samples for the variables whose weight can be interpreted as a relative frequency. The set of samples without the weight, instead, can be interpreted as the prior density of the variable. These two sets of samples can be used to plot the density before and after observing the evidence.

You can sample arguments of queries also for discrete goals: in this case you get a discrete distribution over the values of one or more arguments of a goal.

If the query predicate is determinate in each world, i.e., given values for input arguments there is a single value for output arguments that make the query true, for each sample you get a single value. Moreover, if clauses sharing an atom in the head are mutually exclusive, i.e., in each world the body of at most one clause is true, then the query defines a probability distribution over output arguments. In this way we can simulate those languages such as PRISM and Stochastic Logic Programs [25] that define probability distributions over arguments rather than probability distributions over truth values of ground atoms.

4 Examples

Here we present some examples of PLP in practice. In the first two examples, tile map generation and Markov Logic Networks encoding, all the variables are discrete and no infinite computation path exists.

The next three problems have infinite computation paths: the truel game, the coupon collector problem and the one-dimensional random walk.

The last two examples include continuous variables: latent Dirichlet allocation and the Indian GPA problem.

4.1 Tile map generation

PP and PLP can be used to generate random complex structures. For example, we can write programs for randomly generating maps of video games. We are given a fixed set of tiles that we want to combine to obtain a 2D map that is random but satisfies some soft constraints on the placement of tiles.

Suppose we want to draw a 10x10 map with a tendency to have a lake in the center. The tiles are randomly placed such that, in the central area, water is more probable. The problem can be modeled with the following example[3], where `map(H,W,M)` instantiates `M` to a map of height `H` and width `W`:

```
map(H,W,M):-
  tiles(Tiles),
  length(Rows,H),
  M=..[map,Tiles|Rows],
  foldl(select(H,W),Rows,1,_).

select(H,W,Row,N0,N):-
  length(RowL,W),
  N is N0+1,
  Row=..[row|RowL],
  foldl(pick_row(H,W,N0),RowL,1,_).

pick_row(H,W,N,T,M0,M):-
  M is M0+1,
  pick_tile(N,M0,H,W,T).
```

[3] `http://cplint.ml.unife.it/example/inference/tile_map.swinb`

where `foldl/4` is a SWI-Prolog [58] library predicate that implements the `foldl` meta primitive from functional programming. `pick_tile(Y,X,H,W,T)` returns in `T` a tile for position `(X,Y)` of a map of size `W*H`. The center tile is water:

```
pick_tile(HC,WC,H,W,water):-
  HC is H//2,
  WC is W//2,!.
```

In the central area water is more probable:

```
pick_tile(Y,X,H,W,T):
  discrete(T,[grass:0.05,water:0.9,tree:0.025,rock:0.025]):-
  central_area(Y,X,H,W),!
```

`central_area(Y,X,H,W)` is true if `(X,Y)` is adjacent to the center of the `W*H` map (definition omitted for brevity). In the other places, tiles are chosen at random with distribution `[grass:0.5,water:0.3,tree:0.1,rock:0.1]`:

```
pick_tile(_,_,_,_,T):discrete(T,[grass:0.5,water:0.3,tree:0.1,rock:0.1]).
```

We can generate a map by taking a sample of the query `map(10,10,M)` and collecting the value of `M`. For example, the map of Figure 1 can be obtained[4].

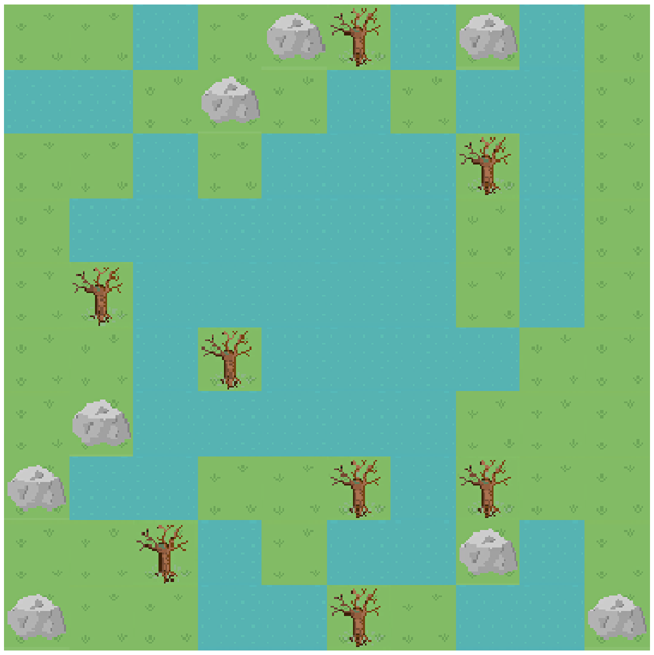
Fig. 1: A random tile map.

[4] Tiles from `https://github.com/silveira/openpixels`

4.2 Markov Logic Networks

Markov Networks (MN) and Markov Logic Networks (MLN) [34] can be encoded with PLP. The encoding is based on the observation that a MN factor can be represented with a Bayesian Network (BN) with an extra node that is always observed. Since PLP programs under the distribution semantics can encode BN [56], we can encode MLN. An example of an MLN clause is

$$1.5\ Intelligent(x) \Rightarrow GoodMarks(x)$$

where 1.5 is the weight of the clause.

For a single constant *anna*, this clause originates an edge between the Boolean nodes for $Intelligent(anna)$ and $GoodMarks(anna)$. This means that the two variables cannot be d-separated in any way. This dependence can be modeled with BN by adding and extra Boolean node, $Clause(anna)$, that is a child of $Intelligent\ (anna)$ and $GoodMarks(anna)$ and is observed. In this way, $Intelligent(anna)$ and $GoodMarks(anna)$ are not d-separated in the BN no matter what other nodes the BN contains.

In general, for a domain with Herbrand base X and an MLN ground clause C mentioning atom variables X', the equivalent BN should contain a Boolean node C with X' as parents. All the query of the form $P(a|b)$ should then be posed to the BN as $P(a|b, C = true)$. The problem is now how to assign values to the conditional probability (CPT) of C given X' so that the joint distribution of X in the BN is the same as that of the MLN.

A ground MLN formulae of the form $\alpha\ C$ contributes to the probabilities of the worlds with a factor e^{α} for the worlds where the clause is true and 1 for the worlds where the clause is false. If we use c to indicate C = true, the joint probability of a state of the world x can then be computed as

$$P(x|c) = \frac{P(x,c)}{P(c)} \propto P(x,c)$$

i.e $P(x|c)$ is proportional to $P(x,c)$, because the denominator does not depend on x and is thus a normalizing constant.

$P(x,c)$ can be written as

$$P(x,c) = P(c|x)P(x) = P(c|x')P(x)$$

where x' is the state of the parents of C, so

$$P(x|c) \propto P(c|x')P(x)$$

To model the MLN formula we just have to ensure that $P(c|x')$ is proportional to e^{α} when x' makes C true and to 1 when x' makes C false. We cannot use e^{α} directly in the CPT for C because it can be larger than 1 but we can use the values $e^{\alpha}/(1+e^{\alpha})$ and $1/(1+e^{\alpha})$ that are proportional to e^{α} and 1 and are surely less than 1.

For an MLN containing the example formula above, the probability of a world would be represented by $P(i, g|c)$ where i and g are values for $Intelligent(anna)$ and $GoodMarks(anna)$ and c is $Clause(anna) = true$. The CPT will have the values $e^{1.5}/(1+e^{1.5})$ for $Clause(anna)$ being true given that the parents' values make the clause true and $1/(1+e^{1.5})$ is the probability of $Clause(anna)$ being true given that the parents' values make the clause false.

In order to model MLN formulas with LPADs, we can add an extra atom $clause_i(X)$ for each formula $F_i = \alpha_i\ C_i$ where X is the vector of variables appearing in C_i. Then, when we query for the probability of query q given evidence e, we have to ask for the probability of q given $e \wedge ce$ where ce is the conjunction of all the groundings of $clause_i(X)$ for all values of i. Then, clause C_i should be transformed into a Disjunctive Normal Form (DNF) formula $C_{i1} \vee \ldots \vee C_{in_i}$ where the disjuncts are mutually exclusive and the LPAD should contain the clauses

$$clause_i(X) : e^{\alpha}/(1+e^{\alpha}) \leftarrow C_{ij}$$

for all j. Similalry, $\neg C_i$ should be transformed into a disjoint sum $D_{i1} \vee \ldots \vee D_{im_i}$ and the LPAD should contain the clauses

$$clause_i(X) : 1/(1+e^{\alpha}) \leftarrow D_{il}$$

for all l.

Alternatively, if α is negative, e^{α} will be smaller than 1 and we can use the two probability values e^{α} and 1 with the clauses

$$\begin{array}{l} clause_i(X) : e^{\alpha} \leftarrow C_{ij} \\ \ldots \\ clause_i(X) \leftarrow D_{il} \end{array}$$

This solution has the advantage that some clauses are certain, reducing the number of random variables. If α is positive in formula $\alpha\ C$, we can consider $-\alpha\ \neg C$.

MLN formulas can also be added to a regular probabilistic logic program. In this case their effect is equivalent to a soft form of evidence, where certain worlds are weighted more than others. This is the same as soft evidence in Figaro [30]. MLN hard constraints, i.e., formulas with an infinite weight, can instead be used to rule out completely certain worlds, those violating the constraint. For example, given hard constraint C equivalent to the sum $C_{i1} \vee \ldots \vee C_{in_i}$, the LPAD should contain the clauses

$$clause_i(X) \leftarrow C_{ij}$$

for all j, and the evidence should contain $clause_i(x)$ for all groundings x of X. In this way, the worlds that violate C are ruled out. Let see an example[5] where we translate the MLN

```
1.5 Intelligent(x) => GoodMarks(x)
1.1 Friends(x, y) => (Intelligent(x) <=> Intelligent(y))
```

[5] `http://cplint.ml.unife.it/example/inference/mln.swinb`

The first MLN formula is translated into

```
clause1(X): 0.8175744762:- \+intelligent(X).
clause1(X): 0.1824255238:- intelligent(X), \+good_marks(X).
clause1(X): 0.8175744762:- intelligent(X), good_marks(X).
```

where $0.8175744762 = e^{1.5}/(1 + e^{1.5})$ and $0.1824255238 = 1/(1 + e^{1.5})$.

The MLN formula

```
1.1 Friends(x, y) => (Intelligent(x) <=> Intelligent(y))
```

is translated into the clauses

```
clause2(X,Y): 0.7502601056:-
  \+friends(X,Y).
clause2(X,Y): 0.7502601056:-
  friends(X,Y), intelligent(X),intelligent(Y).
clause2(X,Y): 0.7502601056:-
  friends(X,Y), \+intelligent(X),\+intelligent(Y).
clause2(X,Y): 0.2497398944:-
  friends(X,Y), intelligent(X),\+intelligent(Y).
clause2(X,Y): 0.2497398944:-
  friends(X,Y), \+intelligent(X),intelligent(Y).
```

where $0.7502601056 = e^{1.1}/(1+e^{1.1})$ and $0.2497398944 = 1/(1+e^{1.1})$. A priori we have a uniform distribution over student intelligence, good marks and friendship:

```
intelligent(_):0.5.
good_marks(_):0.5.
friends(_,_):0.5.
```

and there are two students:

```
student(anna).
student(bob).
```

The evidence must include the truth of all groundings of the $clause_i$ predicates:

```
evidence_mln:- clause1(anna),clause1(bob),clause2(anna,anna),
    clause2(anna,bob),clause2(bob,anna),clause2(bob,bob).
```

We want to query the probability that Anna gets good marks given that she is friend with Bob and Bob is intelligent, so we define

```
ev_intelligent_bob_friends_anna_bob:-
    intelligent(bob),friends(anna,bob),evidence_mln.
```

and query for $P($`good_marks(anna)`$|$`ev_intelligent_bob_friends_anna_bob`$)$ obtaining 0.7330 which is higher than the prior probability 0.6069 of Anna getting good marks, obtained with the query $P($`good_marks(anna)`$|$`evidence_mln`$)$.

4.3 Truel

A truel [22] is a duel among three opponents. There are three truelists, a, b and c, that take turns in shooting with a gun. The firing order is a, b and c. Each truelist can shoot at another truelist or at the sky (deliberate miss). The truelists have these probabilities of hitting the target (if they are not aiming at the sky): 1/3, 2/3 and 1 for a, b and c respectively. The aim for each truelist is to kill all the other truelists. The question is: what should a do to maximize his probability of winning? Aim at b, c or the sky?

Let us see first the strategy for the other truelists and situations. When only two players are left, the best strategy is to shoot at the other player.

When all three players remain, the best strategy for b is to shoot at c, since if c shoots at him he his dead and if c shoots at a, b remains with c which is the best shooter. Similarly, when all three players remain, the best strategy for c is to shoot at b, since in this way he remains with a, the worst shooter.

For a it is more complex. Let us first compute the probability of a to win a duel with a single opponent. When a and c remain, a wins if it shoots c, with probability 1/3. If he misses c, c will surely kill him. When a and b remain, the probability p of a to win can be computed with

$$\begin{aligned}
p &= P(\text{a hits b}) + P(\text{a misses b})P(\text{b misses a})p \\
p &= 1/3 + 2/3 \times 1/3 \times p \\
p &= 3/7
\end{aligned}$$

The probability can be also computed by building the probability tree of Figure 2. The probability that a survives is thus

$$\begin{aligned}
p &= 1/3 + 2/3 \cdot 1/3 \cdot 1/3 + 2/3 \cdot 1/3 \cdot 2/3 \cdot 1/3 \cdot 1/3 + \ldots = \\
&= 1/3 + 2/3^3 + 2^2/3^5 + \ldots = \frac{1}{3} + \sum_{i=0}^{\infty} \frac{2}{3^3}\left(\frac{2}{9}\right)^i = \frac{1}{3} + \frac{\frac{2}{3^3}}{1-\frac{2}{9}} = \\
&= \frac{1}{3} + \frac{\frac{2}{3^3}}{\frac{7}{9}} = \frac{1}{3} + \frac{\frac{2}{3}}{7} = \frac{1}{3} + \frac{2}{21} = \frac{9}{21} = \frac{3}{7}
\end{aligned}$$

When all three players remain, if a shoots at b, b is dead with probability 1/3 but then c will kill a. If b is not dead (probability 2/3), b shoots at c and kills him with probability 2/3. In this case, a is left in a duel with b, with probability of surviving of 3/7. If b doesn't kill c (probability 1/3), c will kill b surely and a is left in a duel with c, with a probability of surviving of 1/3. So overall, if a shoots at b, his probability of winning is

$$2/3 \cdot 2/3 \cdot 3/7 + 2/3 \cdot 1/3 \cdot 1/3 = 4/21 + 2/27 = \frac{36+15}{189} = \frac{50}{189} = 0.2645$$

When all three players remain, if a shoots at c, c is dead with probability 1/3. b then shoots at a and a survives with probability 1/3 and a is then in a duel with b and surviving with probability 3/7. If c survives (probability 2/3), b shoots at

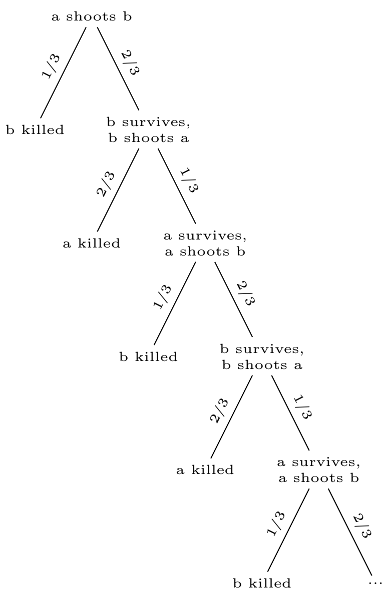

Fig. 2: Probability tree of the truel with opponents a and b.

c and kills him with probability 2/3, so a remains in a duel with b and wins with probability 3/7. If c survives again, he kills b surely and a is left in a duel with c, with probability 1/3 of winning. So overall, if a shoots at c, his probability of winning is

$$1/3{\cdot}1/3{\cdot}3/7+2/3{\cdot}2/3{\cdot}3/7+2/3{\cdot}1/3{\cdot}1/3 = 1/21+4/21+2/27 = 59/189 = 0.3122$$

When all three players remain, if a shoots at the sky, b shoots at c and kills him with probability 2/3, with a remaining in a duel with b. If b doesn't kill c, c will surely kill b and a remains in a duel with c. So overall, if a shoots at the sky, his probability of winning is

$$2/3 \cdot 3/7 + 1/3 \cdot 1/3 = 2/7 + 1/9 = 25/63 = 0.3968$$

This problem can be modeled with an LPAD[6]. However, as can be seen from Figure 2, the number of explanations may be infinite so we have to use an appropriate exact inference algorithm or Monte Carlo inference. We discuss below a program that uses MCINTYRE.

`survives_action(A,L0,T,S)` is true if `A` survives truel performing action `S` with `L0` still alive in turn `T`:

```
survives_action(A,L0,T,S):-
  shoot(A,S,L0,T,L1),
```

[6] `http://cplint.ml.unife.it/example/inference/truel.pl`

```
  remaining(L1,A,Rest),
  survives_round(Rest,L1,A,T).
```

shoot(H,S,L0,T,L) is true when H shoots at S in round T with L0 and L the list of truelists still alive before and after the shot:

```
shoot(H,S,L0,T,L):-
    (S=sky -> L=L0
    ;  (hit(T,H) ->  delete(L0,S,L)
      ;  L=L0
      )
    ).
```

The probabilities of each truelist to hit the chosen target are

```
hit(_,a):1/3.
hit(_,b):2/3.
hit(_,c):1.
```

survives(L,A,T) is true if individual A survives the truel with truelists L at round T:

```
survives([A],A,_):-!.

survives(L,A,T):-
  survives_round(L,L,A,T).
```

survives_round(Rest,L0,A,T) is true if individual A survives the truel at round T with Rest still to shoot and L0 still alive:

```
survives_round([],L,A,T):-
  survives(L,A,s(T)).

survives_round([H|_Rest],L0,A,T):-
  base_best_strategy(H,L0,S),
  shoot(H,S,L0,T,L1),
  remaining(L1,H,Rest1),
  member(A,L1),
  survives_round(Rest1,L1,A,T).
```

These strategies are easy to find:

```
base_best_strategy(b,[b,c],c).
base_best_strategy(c,[b,c],b).
base_best_strategy(a,[a,c],c).
base_best_strategy(c,[a,c],a).
base_best_strategy(a,[a,b],b).
base_best_strategy(b,[a,b],a).
base_best_strategy(b,[a,b,c],c).
base_best_strategy(c,[a,b,c],b).
```

Auxiliary predicate remaining/3 is defined as

```
remaining([A|Rest],A,Rest):-!.

remaining([_|Rest0],A,Rest):-
  remaining(Rest0,A,Rest).
```

We can decide the best strategy for `a` by asking the queries

```
survives_action(a,[a,b,c],0,b)
survives_action(a,[a,b,c],0,c)
survives_action(a,[a,b,c],0,sky)
```

If we take 1000 samples, possible answers are 0.256, 0.316 and 0.389, showing that `a` should aim at the sky.

4.4 Coupon Collector Problem

The coupon collector problem is described in [21] as

> Suppose each box of cereal contains one of N different coupons and once a consumer has collected a coupon of each type, he can trade them for a prize. The aim of the problem is determining the average number of cereal boxes the consumer should buy to collect all coupon types, assuming that each coupon type occurs with the same probability in the cereal boxes.

If there are N different coupons, how many boxes, T, do I have to buy to get the prize? This problem can be modeled by a program[7] defining predicate `coupons/2` such that `coupons(N,T)` is true if we need `T` boxes to get `N` coupons. We represent the coupons with a term for functor `cp/N` with the number of coupons as arity. The ith argument of the term is 1 if the ith coupon has been collected and is a variable otherwise. The term thus represents an array:

```
coupons(N,T):-
  length(CP,N),
  CPTerm=..[cp|CP],
  new_coupon(N,CPTerm,0,N,T).
```

If 0 coupons remain to be collected, the collection ends:

```
new_coupon(0,_CP,T,_N,T).
```

If `N0` coupons remain to be collected, collect one and recurse:

```
new_coupon(N0,CP,T0,N,T):-
  N0>0,
  collect(CP,N,T0,T1),
  N1 is N0-1,
  new_coupon(N1,CP,T1,N,T).
```

`collect/4` collects one new coupon and updates the number of boxes bought:

[7] `http://cplint.ml.unife.it/example/inference/coupon.swinb`

```
collect(CP,N,T0,T):-
  pick_a_box(T0,N,I),
  T1 is T0+1,
  arg(I,CP,CPI),
  (var(CPI)-> CPI=1, T=T1
  ; collect(CP,N,T1,T)
  ).
```

pick_a_box/3 randomly picks a box, an element from the list $[1 \ldots N]$:

```
pick_a_box(_,N,I):uniform(I,L):- numlist(1, N, L).
```

If there are 5 different coupons, we may ask:

– how many boxes do I have to buy to get the prize?
– what is the distribution of the number of boxes I have to buy to get the prize?
– what is the expected number of boxes I have to buy to get the prize?

To answer the first query, we can take a single sample for the query `coupons(5,T)`: in the sample, the query will succeed as `coupons/2` is a determinate predicate and the result will instantiate `T` to a specific value. For example, we may get `T=15`. Note that the maximum number of boxes to buy is unbounded but the case where we have to buy an infinite number of boxes has probability 0, so sampling will surely finish.

To compute the distribution on the number of boxes, we can take a number of samples, say 1000, and plot the number of times a value is obtained as a function of the value. We can do so by dividing the domain of the number of boxes in intervals and counting the number of sampled values that fall in each interval. By doing so we may get the graph in Figure 3.

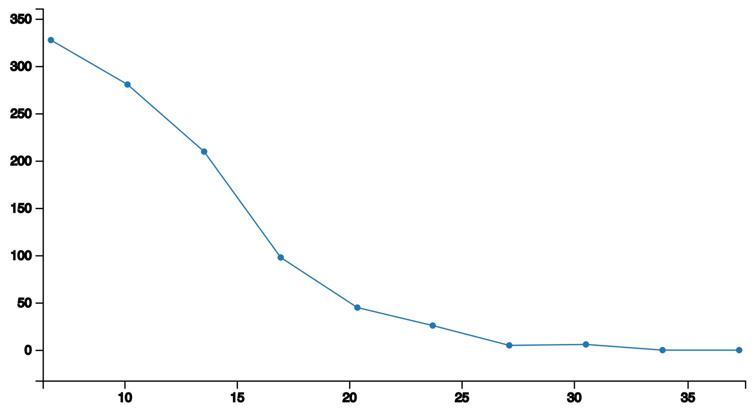

Fig. 3: Distribution of the number of boxes.

To compute the expected number of boxes, we can take a number of samples, say 100, of `coupons(5,T)`. Each sample will instantiate `T`. By summing all these values and dividing the 100, the number of samples, we can get an estimate of the expectation. For example, we may get a value of 11.47.

We can also plot the dependency of the expected number of boxes from the number of coupons, obtaining Figure 4. As observed in [21] , the number of boxes grows as $O(N \log N)$ where N is the number of coupons. The graph shows the accordance of the two curves.

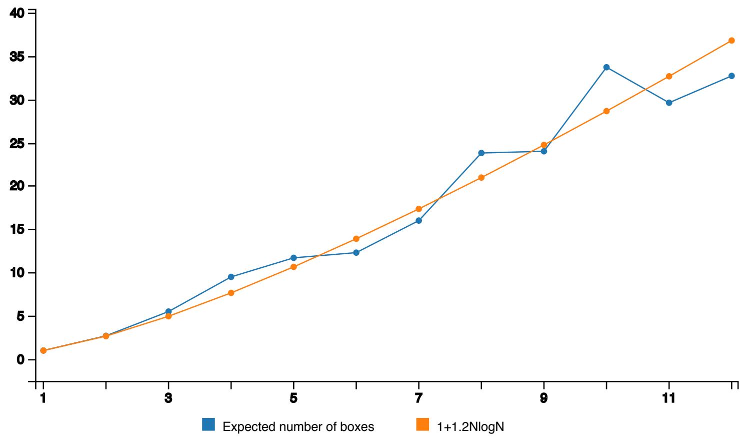

Fig. 4: Expected number of boxes as a function the number of coupons.

The coupon collector problem is similar to the sticker collector problem, where you have an album with a space for every different sticker, you can buy stickers in packs and your objective is to complete the album. A program for the coupon collector problem can be applied to solve the sticker collector problem: if you have N different stickers and packs contain P stickers, we can solve the coupon collector problem for N coupons and get the number of boxes B. Then the number of packs you have to buy to complete the collection is $\lceil B/P \rceil$. So we can write:

```
stickers(N,P,T):- coupons(N,T0), T is ceiling(T0/P).
```

If there are 50 different stickers and packs contain 4 stickers, by sampling the query `stickers(50,4,T)` we can get T=47, i.e., we have to buy 47 packs to complete the entire album.

4.5 One-Dimensional Random Walk

We consider the version of the problem described in [21]: a particle starts at position $x = 10$ and moves with equal probability one unit to the left or one unit to the right in each turn. The random walk stops if the particle reaches position $x = 0$.

The walk terminates with probability one [19] but requires, on average, an infinite time, i.e., the expected number of turns is infinite [21].

We can compute the number of turns with the following program[8]. The walk starts at time 0 and $x = 10$:

```
walk(T):- walk(10,0,T).
```

If x is 0, the walk ends otherwise the particle makes a move:

```
walk(0,T,T).

walk(X,T0,T):-
  X>0,
  move(T0,Move),
  T1 is T0+1,
  X1 is X+Move,
  walk(X1,T1,T).
```

The move is either one step to the left or to the right with equal probability.

```
move(T,1):0.5; move(T,-1):0.5.
```

By sampling the query `walk(T)` we obtain a success as `walk/1` is determinate. The value for `T` represents the number of turns. For example, we may get `T = 3692`.

4.6 Latent Dirichlet Allocation

Text mining [18] aims at extracting knowledge from texts. Latent Dirichlet Allocation (LDA) [6] is a text mining technique which assigns topics from a finite set to words in documents. The model describes a generative process where documents are represented as random mixtures over latent topics and each topic defines a distribution over words. LDA assumes the following generative process for a corpus D consisting of M documents each of length N_i:

1. Choose $\theta_i \sim \mathrm{Dir}(\alpha)$, where $i \in \{1, \ldots, M\}$ and $\mathrm{Dir}(\alpha)$ is the Dirichlet distribution with parameter α
2. Choose $\varphi_k \sim \mathrm{Dir}(\beta)$, where $k \in \{1, \ldots, K\}$
3. For each of the word positions i, j, where $j \in \{1, \ldots, N_i\}$, and $i \in \{1, \ldots, M\}$
   (a) Choose a topic $z_{i,j} \sim \mathrm{Categorical}(\theta_i)$.
   (b) Choose a word $w_{i,j} \sim \mathrm{Categorical}(\varphi_{z_{i,j}})$.

This is a smoothed LDA model to be precise. The subscript is often dropped, as in the plate diagrams 5. The aim is to compute the word probabilities of each topic, the topic of each word, and the particular topic mixture of each document.

[8] `http://cplint.ml.unife.it/example/inference/random_walk.swinb`

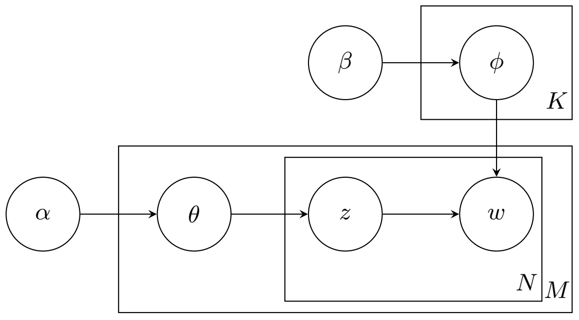

Fig. 5: Smoothed LDA.

This can be done with Bayesian inference: the documents in the dataset represent the observations (evidence) and we want to compute the posterior distribution of the above quantities.

This problem can modeled by the MCINTYRE program[9] below, where predicate

```
word(Doc,Position,Word)
```

indicates that document `Doc` in position `Position` (from 1 to the number of words of the document) has word `Word` and predicate

```
topic(Doc,Position,Topic)
```

indicates that document `Doc` associates topic `Topic` to the word in position `Position`. We also assume that the distributions for both θ_m and φ_k are symmetric Dirichlet distributions with scalar concentration parameter η set using a fact for the predicate `eta/1`, i.e., $\alpha = \beta = [\eta, \ldots, \eta]$. The program is then:

```
theta(_,Theta):dirichlet(Theta,Alpha):-
  alpha(Alpha).

topic(DocumentID,_,Topic):discrete(Topic,Dist):-
  theta(DocumentID,Theta),
  topic_list(Topics),
  maplist(pair,Topics,Theta,Dist).

word(DocumentID,WordID,Word):discrete(Word,Dist):-
  topic(DocumentID,WordID,Topic),
  beta(Topic,Beta),
  word_list(Words),
  maplist(pair,Words,Beta,Dist).

beta(_,Beta):dirichlet(Beta,Parameters):-
  n_words(N),
```

[9] `http://cplint.ml.unife.it/example/inference/lda.swinb`

```
  eta(Eta),
  findall(Eta,between(1,N,_),Parameters).

alpha(Alpha):-
  eta(Eta),
  n_topics(N),
  findall(Eta,between(1,N,_),Alpha).

eta(2).

pair(V,P,V:P).
```

where `maplist/4` is a library of SWI-Prolog encoding the `maplist` primitive of functional programming. Suppose we have two topics, indicated with integers 1 and 2, and 10 words, indicated with integers $1, \ldots, 10$:

```
topic_list(L):-
  n_topics(N),
  numlist(1,N,L).

word_list(L):-
  n_words(N),
  numlist(1,N,L).

n_topics(2).

n_words(10).
```

We can, for example, use the model generatively and sample values for word in position 1 of document 1. The histogram of the frequency of word values when taking 100 samples is shown in Figure 6.

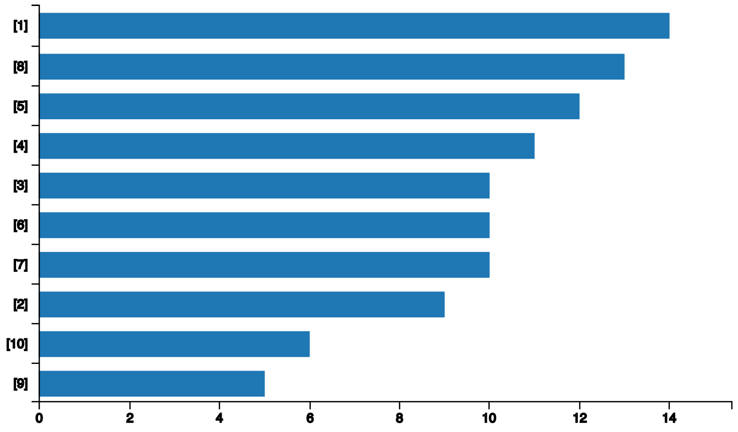

Fig. 6: Values for word in position 1 of document 1.

We can also sample values for couples (word, topic) in position 1 of document 1. The histogram of the frequency of the couples when taking 100 samples is shown in Figure 7.

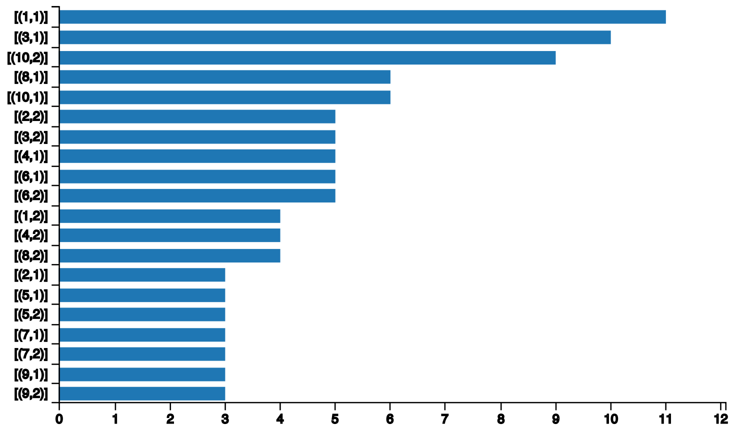

Fig. 7: Values for couples (word,topic) in position 1 of document 1.

We can use the model to classify the words into topics. Here we use conditional inference with Metropolis-Hastings that is implemented in MCINTYRE. A priori both topics are about equally probable for word 1 of document , so if we take 100 samples of `topic(1,1,T)` we get the histogram in Figure 8. If we

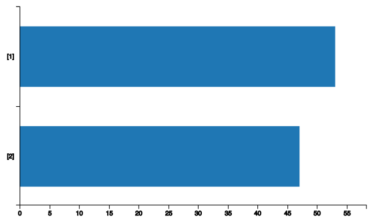

Fig. 8: Prior distribution of topics for word in position 1 of document 1.

observe that words 1 and 2 of document 1 are equal (`word(1,1,1)`,`word(1,2,1)` as evidence) and take again 100 samples, one of the topics gets more probable, as the histogram of Figure 9 shows. You can also see this if you look at the density of the probability of topic 1 before and after observing that words 1 and 2 of document 1 are equal: the observation makes the distribution less uniform, see Figure 10

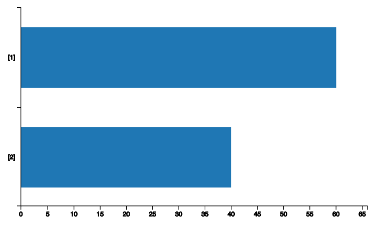

Fig. 9: Posterior distribution of topics for word in position 1 of document 1.

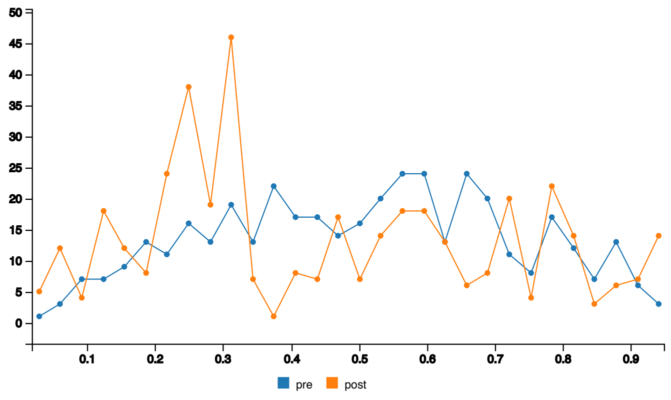

Fig. 10: Density of the probability of topic 1 before and after observing that words 1 and 2 of document 1 are equal.

4.7 The Indian GPA Problem

In the Indian GPA problem proposed by Stuart Russel [29,28] the question is: if you observe that a student GPA is exactly 4.0, what is the probability that the student is from India, given that the American GPA score is from 0.0 to 4.0 and the Indian GPA score is from 0.0 to 10.0? Stuart Russel observed that most probabilistic programming system are not able to deal with this query because it requires combining continuous and discrete distributions. This problem can be modeled by building a mixture of a continuous and a discrete distribution for each nation to account for grade inflation (extreme values have a non-zero probability). Then the probability of the student's GPA is a mixture of the nation mixtures. From statistics, given this model and the fact that the student's GPA is exactly 4.0, the probability that the student is American must be 1.0.

This problem can be modeled with Anglican, DC and MCINTYRE. In MCINTYRE we can model it with the program below[10]. The probability distribution of GPA scores for American students is continuous with probability 0.95 and discrete with probability 0.05:

```
is_density_A:0.95;is_discrete_A:0.05.
```

[10] http://cplint.ml.unife.it/example/inference/indian_gpa.pl

The GPA of an American student follows a beta distribution if the distribution is continuous:

```
agpa(A): beta(A,8,2) :- is_density_A.
```

The GPA of an American student is 4.0 with probability 0.85 and 0.0 with probability 0.15 if the distribution is discrete:

```
american_gpa(G) : finite(G,[4.0:0.85,0.0:0.15]) :- is_discrete_A.
```

or is obtained by rescaling the value of returned by `agpa/1` to the (0.0,4.0) interval:

```
american_gpa(A):- agpa(A0), A is A0*4.0.
```

The probability distribution of GPA scores for Indian students is continuous with probability 0.99 and discrete with probability 0.01.

```
is_density_I : 0.99; is_discrete_I:0.01.
```

The GPA of an Indian student follows a beta distribution if the distribution is continuous:

```
igpa(I): beta(I,5,5) :- is_density_I.
```

The GPA of an Indian student is 10.0 with probability 0.9 and 0.0 with probability 0.1 if the distribution is discrete:

```
indian_gpa(I): finite(I,[0.0:0.1,10.0:0.9]):-  is_discrete_I.
```

or is obtained by rescaling the value returned by `igpa/1` to the (0.0,10.0) interval:

```
indian_gpa(I) :- igpa(I0), I is I0*10.0.
```

The nation is America with probability 0.25 and India with probability 0.75.

```
nation(N) : finite(N,[a:0.25,i:0.75]).
```

The GPA of the student is computed depending on the nation:

```
student_gpa(G) :- nation(a),american_gpa(G).
student_gpa(G) :- nation(i),indian_gpa(G).
```

If we query the probability that the nation is America given that the student got 4.0 in his GPA we obtain 1.0, while the prior probability that the nation is America is 0.25.

5 Conclusions

PLP has now become mature enough to encode and solve a wide variety of problems. The recent inclusion of programs with infinite computation paths and continuous random variables closed the gap with other PP formalism, making PLP a valid alternative.

We have presented an overview of the semantics and of the main inference approaches, together with a set of examples that we believe show the maturity of the field.

Online tutorials on PLP are available at `http://ds.ing.unife.it/~gcota/plptutorial/` [42] and `https://dtai.cs.kuleuven.be/problog/tutorial.html`.

References

1. Alberti, M., Bellodi, E., Cota, G., Riguzzi, F., Zese, R.: cplint on SWISH: Probabilistic Logical Inference with a Web Browser. Intell. Artif. 11(1), 47–64 (2017)
2. Bellodi, E., Lamma, E., Riguzzi, F., Costa, V.S., Zese, R.: Lifted variable elimination for probabilistic logic programming. Theor. Pract. Log. Prog. 14(4-5), 681–695 (2014)
3. Bellodi, E., Riguzzi, F.: Experimentation of an Expectation Maximization algorithm for probabilistic logic programs. Intell. Artif. 8(1), 3–18 (2012)
4. Bellodi, E., Riguzzi, F.: Expectation Maximization over Binary Decision Diagrams for probabilistic logic programs. Intell. Data Anal. 17(2), 343–363 (2013)
5. Bellodi, E., Riguzzi, F.: Structure learning of probabilistic logic programs by searching the clause space. Theor. Pract. Log. Prog. 15(2), 169–212 (2015)
6. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet allocation. J. Mach. Learn. Res. 3, 993–1022 (2003)
7. Cozman, F.G., Mauá, D.D.: The structure and complexity of credal semantics. In: Hommersom, A., Abdallah, S.A. (eds.) PLP 2016. CEUR Workshop Proceedings, vol. 1661, pp. 3–14. CEUR-WS.org (2016)
8. Dantsin, E.: Probabilistic logic programs and their semantics. In: Russian Conference on Logic Programming. LNCS, vol. 592, pp. 152–164. Springer (1991)
9. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. 17, 229–264 (2002)
10. De Raedt, L., Kersting, K.: Probabilistic inductive logic programming. In: Ben-David, S., Case, J., Maruoka, A. (eds.) ALT 2004. LNCS, vol. 3244, pp. 19–36. Springer (2004)
11. De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Mach. Learn. 100(1), 5–47 (2015)
12. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: Veloso, M.M. (ed.) IJCAI 2007. vol. 7, pp. 2462–2467. AAAI Press, Palo Alto, California USA (2007)
13. Di Mauro, N., Bellodi, E., Riguzzi, F.: Bandit-based Monte-Carlo structure learning of probabilistic logic programs. Mach. Learn. 100(1), 127–156 (2015)
14. Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D.S., Gutmann, B., Thon, I., Janssens, G., De Raedt, L.: Inference and learning in probabilistic logic programs using weighted boolean formulas. Theor. Pract. Log. Prog. 15(3), 358–401 (2015)
15. Gorlin, A., Ramakrishnan, C.R., Smolka, S.A.: Model checking with probabilistic tabled logic programming. Theor. Pract. Log. Prog. 12(4-5), 681–700 (2012)
16. Gutmann, B., Thon, I., Kimmig, A., Bruynooghe, M., De Raedt, L.: The magic of logical inference in probabilistic programming. Theor. Pract. Log. Prog. 11(4-5), 663–680 (2011)
17. Holzinger, A.: Introduction to machine learning and knowledge extraction (MAKE). Mach. Learn. Knowl. Extr. 1(1), 1–20 (2017)
18. Holzinger, A., Schantl, J., Schroettner, M., Seifert, C., Verspoor, K.: Biomedical text mining: State-of-the-art, open problems and future challenges. In: Holzinger, A., Jurisica, I. (eds.) Interactive Knowledge Discovery and Data Mining in Biomedical Informatics, LNCS, vol. 8401, pp. 271–300. Springer (2014)
19. Hurd, J.: A formal approach to probabilistic termination. In: Carreño, V., Muñoz, C.A., Tahar, S. (eds.) TPHOLs 2002. LNCS, vol. 2410, pp. 230–245. Springer (2002)

20. Islam, M.A., Ramakrishnan, C., Ramakrishnan, I.: Inference in probabilistic logic programs with continuous random variables. Theor. Pract. Log. Prog. 12, 505–523 (2012)
21. Kaminski, B.L., Katoen, J.P., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected run—times of probabilistic programs. In: Thiemann, P. (ed.) ESOP 2016. LNCS, vol. 9632, pp. 364–389. Springer (2016)
22. Kilgour, D.M., Brams, S.J.: The truel. Mathematics Magazine 70(5), 315–326 (1997)
23. Kimmig, A., Demoen, B., De Raedt, L., Costa, V.S., Rocha, R.: On the implementation of the probabilistic logic programming language ProbLog. Theor. Pract. Log. Prog. 11(2-3), 235–262 (2011)
24. Meert, W., Struyf, J., Blockeel, H.: CP-Logic theory inference with contextual variable elimination and comparison to BDD based inference methods. In: De Raedt, L. (ed.) ILP 2009. LNCS, vol. 5989, pp. 96–109. Springer (2010)
25. Muggleton, S.: Learning stochastic logic programs. Electron. Trans. Artif. Intell. 4(B), 141–153 (2000)
26. Nampally, A., Ramakrishnan, C.: Adaptive MCMC-based inference in probabilistic logic programs. arXiv preprint arXiv:1403.6036 (2014)
27. Ng, R.T., Subrahmanian, V.S.: Probabilistic logic programming. Inf. Comput. 101(2), 150–201 (1992)
28. Nitti, D., De Laet, T., De Raedt, L.: Probabilistic logic programming for hybrid relational domains. Mach. Learn. 103(3), 407–449 (2016)
29. Perov, Y., Paige, B., Wood, F.: The Indian GPA problem (2017), `http://www.robots.ox.ac.uk/~fwood/anglican/examples/viewer/?worksheet=indian-gpa`, accessed April 15, 2017
30. Pfeffer, A.: Practical Probabilistic Programming. Manning Publications (2016)
31. Poole, D.: The Independent Choice Logic for modelling multiple agents under uncertainty. Artif. Intell. 94, 7–56 (1997)
32. Poole, D.: Abducing through negation as failure: stable models within the independent choice logic. J. Logic Program. 44(1-3), 5–35 (2000)
33. Poole, D.: Probabilistic horn abduction and Bayesian networks. Artif. Intell. 64(1), 81–129 (1993)
34. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. 62(1-2), 107–136 (2006)
35. Riguzzi, F.: ALLPAD: Approximate learning of logic programs with annotated disjunctions. Mach. Learn. 70(2-3), 207–223 (2008)
36. Riguzzi, F.: MCINTYRE: A Monte Carlo system for probabilistic logic programming. Fund. Inform. 124(4), 521–541 (2013)
37. Riguzzi, F.: Speeding up inference for probabilistic logic programs. Comput. J. 57(3), 347–363 (2014)
38. Riguzzi, F.: The distribution semantics for normal programs with function symbols. Int. J. Approx. Reason. 77, 1 – 19 (2016)
39. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., Cota, G.: Probabilistic logic programming on the web. Softw.-Pract. Exper. 46(10), 1381–1396 (October 2016)
40. Riguzzi, F., Bellodi, E., Zese, R., Cota, G., Lamma, E.: Scaling structure learning of probabilistic logic programs by mapreduce. In: Fox, M., Kaminka, G. (eds.) ECAI 2016. vol. 285, pp. 1602–1603. IOS Press (2016)
41. Riguzzi, F., Bellodi, E., Zese, R., Cota, G., Lamma, E.: A survey of lifted inference approaches for probabilistic logic programming under the distribution semantics. Int. J. Approx. Reason. 80, 313–333 (January 2017)

42. Riguzzi, F., Cota, G.: Probabilistic logic programming tutorial. The Association for Logic Programming Newsletter 29(1), 1–1 (March/April 2016)
43. Riguzzi, F., Di Mauro, N.: Applying the information bottleneck to statistical relational learning. Mach. Learn. 86(1), 89–114 (2012)
44. Riguzzi, F., Lamma, E., Alberti, M., Bellodi, E., Zese, R., Cota, G.: Probabilistic logic programming for natural language processing. In: Chesani, F., Mello, P., Milano, M. (eds.) Workshop on Deep Understanding and Reasoning, URANIA 2016. CEUR Workshop Proceedings, vol. 1802, pp. 30–37 (2017)
45. Riguzzi, F., Swift, T.: The PITA system: Tabling and answer subsumption for reasoning under uncertainty. Theor. Pract. Log. Prog. 11(4–5), 433–449 (2011)
46. Riguzzi, F., Swift, T.: Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics. Theor. Pract. Log. Prog. 13(Special Issue 02 - 25th Annual GULP Conference), 279–302 (2013)
47. Riguzzi, F., Zese, R., Cota, G.: Probabilistic inductive logic programming on the web. In: EKAW 2016, LNCS, vol. 10180, pp. 172–175. Springer (2017)
48. Sato, T., Meyer, P.: Infinite probability computation by cyclic explanation graphs. Theor. Pract. Log. Prog. 14, 909–937 (11 2014)
49. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) ICLP 1995. pp. 715–729. MIT Press, Cambridge, Massachusetts (1995)
50. Sato, T., Kameya, Y.: PRISM: a language for symbolic-statistical modeling. In: IJCAI 1997. vol. 97, pp. 1330–1339 (1997)
51. Sato, T., Kubota, K.: Viterbi training in prism. Theor. Pract. Log. Prog. 15(02), 147–168 (2015)
52. Sato, T., Meyer, P.: Tabling for infinite probability computation. In: Dovier, A., Costa, V.S. (eds.) ICLP TC 2012. LIPIcs, vol. 17, pp. 348–358 (2012)
53. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM J. Comput. 8(3), 410–421 (1979)
54. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM 38(3), 620–650 (1991)
55. Vennekens, J., Verbaeten, S.: Logic programs with annotated disjunctions. Tech. Rep. CW386, KU Leuven (2003)
56. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3131, pp. 195–209. Springer, Berlin (2004)
57. Von Neumann, J.: Various techniques used in connection with random digits. Nat. Bureau Stand. Appl. Math. Ser. 12, 36–38 (1951)
58. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theor. Pract. Log. Prog. 12(1-2), 67–96 (2012)