Probabilistic Logic in Dynamic Domains: Particle Filter with Distributional Clauses

Davide Nitti, Guy Van den Broeck, and Luc De Raedt

Department of Computer Science, Katholieke Universiteit Leuven,
Celestijnenlaan 200A - bus 2402, 3001 Heverlee, Belgium
{firstname.lastname}@cs.kuleuven.be

Abstract. We introduce a probabilistic logic programming framework to handle continuous distributions as well as dynamic domains for use in fields like robotics. The framework is based on the recently introduced notion of distributional clauses, an extension of Sato's distribution semantics. The key contribution of this paper is the introduction of a particle filter for this formalism. The particle filter recursively updates its beliefs about the current state, where states correspond to logical interpretations and may contain continuous variables. A further contribution of this work is extending distributional clauses with stratification to support negation.

Keywords: probabilistic logic programming, particle filter, robotics

1 Introduction

Probabilistic programming languages and statistical relational learning techniques have proven successful in many application areas ranging from natural language processing to bioinformatics. However, in order to deal with fields such as robotics or vision, they should not only be able to deal with high-level symbolic information but also with low-level numeric information as well as with dynamics. This paper extends probabilistic logic programming techniques to deal with this type of application. The approach is based on the recently introduced distributional clauses [5], which extend Hybrid Problog [4] to be able to represent and reason about continuous distributions. The semantics was rigorously proven along the lines of Sato's distribution semantics [7]. Furthermore we extend distributional clauses with stratification to support negation. The resulting formalism is related to that of other probabilistic programming languages such as BLOG [6], Church [3] in that it allows for dealing with continuous distributions. However, distributional clauses are extended here to deal with dynamic domains, where the state of the environment changes over time. More specifically, we introduce a particle filter that allows to recursively estimate the state the agent is in. Particle filters are widely applied in probabilistic robotics, and therefore, we hope that their incorporation into a probabilistic logic programming formalism will contribute towards the application of relational learning to robotics.

2 Distributional Clauses

In this paper we employ a probabilistic logic language, which consists besides normal Horn clauses of distributional clauses [5]:

Definition 1 (Distributional clause). *A* distributional clause *is a definite clause with an atom* $\mathtt{h} \sim \mathcal{D}$ *in the head where* $\sim$ *is a binary predicate used in infix notation. For each ground instance* $(\mathtt{h} \sim \mathcal{D} \text{ :- } \mathtt{b_1}, \ldots, \mathtt{b_n})\theta$ *with* θ *a substitution over the Herbrand universe of the logic program, the distributional clause defines a random variable* $\mathtt{h}\theta$ *and an associated distribution* $\mathcal{D}\theta$*.*

These random variables are considered as terms of the Herbrand universe. They can be used as any other term in the logic program. Furthermore, a term $\simeq(d)$ with non-Herbrand semantics constructed from the reserved functor $\simeq/1$ represents the outcome of the random variable d.

Example 1 (Distributional clauses).

$$\mathtt{npeople} \sim \mathtt{poisson}(\mathtt{6}). \tag{1}$$

$$\mathtt{position}(\mathtt{P}) \sim \mathtt{uniform}(\mathtt{1}, \mathtt{10}) \text{ :- } \mathtt{between}(\mathtt{1}, \simeq(\mathtt{npeople}), \mathtt{P}). \tag{2}$$

The distributional clause (1) states that the number of people is governed by a Poisson distribution with mean 6. The distributional clause (2) models the position of each person `position(P)` as a random variable uniformly distributed from 1 to 10. Note that the distribution is defined only for the values P for which $between(1, \simeq(npeople), P)$ succeeds.

Inference in distributional clauses is performed by sampling possible worlds (Herbrand models) of the logic program using forward reasoning until a fixed point is reached. Therefore the probability of a query given evidence $p(q|e)$ is estimated as the number of worlds sampled that satisfy the query and evidence divided by the number of worlds that satisfy evidence. To improve the performance of this rejection sampling approach to inference, the inference procedure for distributional clauses outlined in [5] uses magic sets to generate only facts relevant to evaluate the query, and heuristic lookahead to remove inconsistent values (for the evidence) from random variable distributions. This reduces the number of interpretations that have to be rejected due to their inconsistency with the evidence. The reader is referred to [4, 5] for more details.

3 Distributional Clauses over time

To model dynamic domains, we can define a logic program with distributional clauses that describes how the environment evolves over time. Inference in dynamic domains rapidly becomes intractable when using forward sampling, especially using continuous distributions and with many facts as evidence. To evaluate the probability of a query at time t, the inference procedure needs to generate samples starting from time 0. Furthermore the samples should satisfy

the evidence. With increasing t, the sample rejection rate (due to the inconsistency with the evidence) becomes unacceptable even when heuristic lookahead is used. To perform efficient inference over time we need a different method. We therefore propose a particle filter for distributional clauses. The particle filter is recursive over time, which means that the distribution of a state at time t is estimated from the distribution at the previous time $t-1$ and the last observation.

3.1 Classical Particle Filter

One of the most common methods in robotics to model a dynamic environment is the Hidden Markov Model (HMM), where the state of the world is not observable but it influences what the system observes (e.g. through sensors). The state is described as a Markov chain, and one of the inference tasks is to estimate the state given the observations received from the sensors, that is the probability $p(x_t|z_{1:t}) = bel(x_t)$ called belief, where x_t is the current state and $z_{1:t}$ is the set of observations from time 1 to t. A general approach to estimate the (hidden) state over time from observations is the Bayes filter [8]. This algorithm works recursively computing the belief at time t starting from the belief at $t-1$ and the last observation z_t:

$$bel(x_t) = \eta\ p(z_t|x_t) \int p(x_t|x_{t-1}) bel(x_{t-1}) dx_{t-1}$$

where η is a normalization constant. A very popular example of a Bayes filter is the particle filter [8] that uses samples (called particles) to describe the belief distribution. The particle filter can handle non-linear processes and measurement models but it is only an approximation. However with a sufficiently large number of particles the results are reliable. The particle filter takes as input a set of particles of the previous time step: $x_{t-1}^{[1]}, x_{t-1}^{[2]}, ..., x_{t-1}^{[n]}$ distributed as $bel(x_{t-1}) = p(x_{t-1}|z_{1:t-1})$ and the last observation z_t, and it produces a set of particles approximately distributed as $bel(x_t) = p(x_t|z_{1:t})$.
The algorithm can by defined in three steps:

- Sample each particle $x_t^{[m]}$ from a proposal distribution $q(x_t|x_{t-1}^{[m]}, z_t)$
- Assign to each particle the weight $w_t^{[m]} = \frac{p(z_t|x_t^{[m]})p(x_t|x_{t-1}^{[m]})}{q(x_t|x_{t-1}^{[m]}, z_t)}$
- Resample with replacement from the particle set, where the probability of sampling $x_t^{[m]}$ is proportional to $w_t^{[m]}$

If the proposal distribution $q(x_t|x_{t-1}^{[m]}, z_t)$ is $p(x_t|x_{t-1}^{[m]})$ (state transition probability), the weights become $w_t^{[m]} = p(z_t|x_t^{[m]})$. In summary, the particle filter updates in every step the belief at time t $bel(x_t)$ starting from the previous belief and the last observation, and the distribution of the particles is used to represent the belief. In robotics the state x_t usually is a vector of real numbers (e.g., the position and velocity of the robot).

3.2 Particle Filter with Distributional Clauses

We now propose a particle filter for probabilistic logic programs with distributional clauses that encode a state transition model and measurement probability. The distributional program used by the particle filter must contain:

background knowledge consisting of facts and (distributional) clauses that do not change over time

state transition model representing $p(x_t|x_{t-1})$, which defines how the next state can be obtained from the current state using (distributional) clauses with head *next*(*fact*) and literals *current*(*fact*) in the body that pertain to the current state.

prior distribution representing $p(x_0)$ by (distributional) clauses with head *current*(*fact*) and *timestep*(0) in the body. These clauses are activated only for the first timestep.

measurement probability defined as a set of clauses with head *probability*(Obs, P) where P represents the probability of the observation *Obs* given the state: $p(z_t|x_t)$. This predicate defines the measurement probability explicitly and does not infer it.

The proposed particle filter employs the same steps as the classical particle filter, using the state transition probability as proposal distribution. The main differences are that particles represent interpretations and they are sampled from the distributional program.

The algorithm starts by creating a set of particles distributed as $p(x_0)$ by forward sampling from the clauses with *timestep*(0) in the body. Each particle will contain facts *current*(*fact*) that define the state at time 0. Then, the particles for timestep t are generated from the particles of timestep $t-1$. This is done by forward sampling facts *next*(*fact*) (the new state) from *current*(*fact*) (the old state), according to the state transition model defined in the distributional program. These new particles are weighed by their measurement probability, which is computed by evaluating *probability*(*observation*, P) given the new state defined in the particle. Afterwards, the particles are resampled with probability proportional to their weights. Finally the facts *current*(*fact*) and *probability*(*observation*, P) are removed because they referred to the old state, and *next*(*fact*) are changed in *current*(*fact*). Therefore a new belief update can start for the next timestep, now ignoring the clauses with *timestep*(0) in the body.

For example, we can consider people that move in one direction (for simplicity), the background knowledge can be the number of people (1). Instead the prior distribution $p(x_0)$ could defines the initial position and velocity of each person:

$$\texttt{current(pos(P))} \sim \texttt{uniform(1, 10)} \texttt{ :- between(1, } \simeq\texttt{(npeople), P), timestep(0).}$$

$$\texttt{current(vel(P))} \sim \texttt{uniform(}-\texttt{1, 1)} \texttt{ :- between(1, } \simeq\texttt{(npeople), P), timestep(0).}$$

We can define a relation near(P1,P2), that is true when the squared distance between P1 and P2 is less than or equal to 4:

```
current(near(P1,P2)) :-
    dist_leq((≃(current(pos(P1)))− ≃(current(pos(P2))))^2, 4),
    P1\ = P2
```

We use the special predicate `dist_leq/2` (less than or equal to) to compare the outcome of random variables with a value. The state transition model in this example defines the next position and the next velocity of each person P, given the current position and velocity:

```
next(pos(P)) ∼ gaussian(≃(current(pos(P)))+ ≃(current(vel(P))), 0.1).
next(vel(P)) ∼ gaussian(≃(current(vel(P))), 0.05).
```

In the next position and the next velocity we consider gaussian noise.
The measurement probability function should be defined explicitly, for example:

```
probability(sensor(Value),P) :- [...], P is [...].
```

Where the observation `sensor(Value)` has a probability `P` defined in the clause.

4 Negation and stratification

A further contribution of this work is extending distributional clauses [5] with stratification [1] to support the negation of atoms in the body of a (distributional) clause. We implemented stratification in distributional clauses following the classical definition of stratification. To perform the forward reasoning in a stratified program we need to reach a fixed point for each stratum of the program, from low number to high number. Currently the user must assign the stratification number to each (distributional) clause, however it would be straightforward to do this automatically.

5 Experiments

We implemented distributional clauses and the particle filter in YAP Prolog, and performed some experiments to evaluate its performance. The particle filter is generally used in real time, so time performance is a crucial aspect. The method proposed needs to infer facts from a logic program for each particle and for each time step; furthermore to have acceptable results we need a large enough number of particles (the size is related to the dimensionality of the state space). Tests using distributional clauses [5] show that the use of magic sets leads to significant performance improvements and these results are confirmed in the particle filter. We tested the particle filter with a logic program that consists of 8 clauses to define the state transition, 4 clauses to define the measurement probability and 7 clauses for background knowledge and the initial state. The particle filter with distributional clauses was executed on a laptop Core 2 Duo 2 GHz, the runtime is about 0.85 seconds for each time step with 1000 particles and 0.41 seconds for 500 particles. Obviously the performance depends on the logic program complexity, thus for real time applications the program should be as compact as possible.

6 Related Work and Conclusion

Several extensions to the particle filter algorithm have been proposed that deal with logical and relational domains, in a more restricted setting. Zettlemoyer et al. [9] proposed a relational particle filter for logical languages with only discrete random variables, which cannot deal with the continuous distributions we describe here. Recent work on relational dynamic Bayesian networks [2] separates objects (that may have continuous attributes) from relations that hold amongst them.

We propose an unified and powerful approach that encodes the entire environment in one distributional program, including the state transition model and the measurement probability in the form of distributional clauses. We have contributed an extension of distributional clauses for dynamic domains and proposed a particle filter for the resulting probabilistic language. The performance seems acceptable and further code improvement can decrease the runtime allowing to use the particle filter in real-time applications. Finally, with regard to future work, the proposed particle filter could be improved state-of-art techniques such as Rao-Blackwellized Particle Filters or Mixture Particle Filters.

References

[1] Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. In: Foundations of Deductive Databases and Logic Programming., pp. 89–148. Morgan Kaufmann (1988)

[2] Cristina E. Manfredotti David J. Fleet, Howard J. Hamilton, S.Z.: Relational particle filtering. Monte Carlo Methods for Modern Applications, 2010 NIPS Workshop, Whistler, B.C. (December 2010)

[3] Goodman, N., Mansinghka, V.K., Roy, D.M., Bonawitz, K., Tenenbaum, J.B.: Church: A language for generative models. In: UAI. pp. 220–229 (2008)

[4] Gutmann, B., Jaeger, M., De Raedt, L.: Extending ProbLog with continuous distributions. In: Frasconi, P., Lisi, F.A. (eds.) Proceedings of the 20th International Conference on Inductive Logic Programming (ILP–10). Firenze, Italy (2010)

[5] Gutmann, B., Thon, I., Kimmig, A., Bruynooghe, M., De Raedt, L.: The magic of logical inference in probabilistic programming. Theory and Practice of Logic Programming (2011)

[6] Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D., Kolobov, A.: BLOG: Probabilistic models with unknown objects. In: IJCAI. pp. 1352–1359 (2005)

[7] Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Proceedings of the Twelth International Conference on Logic Programming (ICLP 1995). pp. 715–729. MIT Press (1995)

[8] Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press (2005)

[9] Zettlemoyer, L.S., Pasula, H.M., Kaelbling, L.P.: Logical particle filtering. In: In Proceedings of the Dagstuhl Seminar on Probabilistic, Logical, and Relational Learning (2007)