

# Probabilistic Programming for Planning Problems.

Ingo Thon and Bernd Gutmann and Guy Van den Broeck

## Abstract

Probabilistic programming is an emerging field at the intersection of statistical learning and programming languages. An appealing property of probabilistic programming languages (PPL) is their support for constructing arbitrary probability distributions. This allows one to model many different domains and solve a variety of problems. We show the link between probabilistic planning and PPLs by introducing a translation that allows one to map probabilistic planning problems onto parameter learning in PPLs. The advantage of our approach is twofold. Firstly, having the expressivity of a programming language simplifies modeling compared to using existing planning languages such as PPDDL. Secondly, there exist effective general-purpose learning algorithms that – having the correct encoding – can readily be used to learn optimal policies. In this paper we use ProbLog – a probabilistic version of Prolog – as programming language, but our approach can be applied on any other PPL as well.

## Introduction

Planning is one of the oldest problems in artificial intelligence, and yet it is still a very active field of research. This is partly due to the very general definition of planning, but also due to the fact that modern computers are fast enough to solve large real world problems. In this paper, we show how planning problems can be represented using the probabilistic programming language (PPL) ProbLog. The use of a PPL is appealing because it can represent domains having more or less arbitrary properties. Modeling of partial observability, random processes of the environment, concurrent actions, with different and even conflicting goals, fit naturally. Afterwards, we show how to solve the planning problem using existing methods, although not the ones expected. Finally, we discuss further improvements to the existing methods to specialize them for the planning task.

## Planning

A very basic definition of planning is: **given** a set of actions, an initial state and a set of desired goal states, **find** a sequence of actions such that performing this sequence starting in the initial state will reach the goal state. In this basic setting many algorithms that can find satisfying plans have

been developed. Many instances of planning problems can be solved in reasonable time. But classical planning makes the crucial assumptions that **(a)** the domain model is specified in advance and **(b)** the actions are deterministic. This encourages us to push boundaries and move to more general planning problems. Probabilistic planning, for example, is situated in domains where the outcome of an action is not completely determined but follows a certain distribution, violating assumption **(b)**. Planning problems violating **(a)** were formalized within model-lite planning (Yoon and Kambhampati 2007), which does not rely on a complete model of the domain. This is slightly different from *Reinforcement Learning* in that it **(i)** does not consider utilities or costs, but just sets of possible goal states, it **(ii)** assumes that the structure of the domain is known and **(iii)** that the agent has to achieve the goal exactly once. To tackle domains with costs and utilities, Van den Broeck *et al.* (2010) recently developed DTProbLog, a decision theoretic extension of ProbLog. W.r.t. requirement **(ii)**, we believe that in most domains, it is easy to define the preconditions and possible effects of an action, but hard to specify the exact probability distributions in the model. Some existing probabilistic planners like FF-Replan (FFR) (Yoon, Fern, and Givan 2007), typically violate requirement **(iii)**. They might choose actions where a deadlock state might be the outcome like illustrated in the following. Simply speaking, FFR (*all-outcomes determinization*) replaces the action operators by a set of deterministic ones, which are used to find a plan using FF. During execution, the system monitors the state and whenever the result is unexpected, it re-plans - hence the name. This approach has an obvious shortcoming not represented in the IPC planning competition domains, which are the standard benchmark. Consider a domain with two actions: one leading to the goal with above average probability and otherwise killing the agent, one having a high probability that the agent stays in the current state but otherwise reaching the goal state. After applying the determinization approach the agent would blindly select the most dangerous action. Naive approaches to solve this problem, such as altering the action operators or hindsight optimization might require complicated inference. This is due to the fact that the deadlock is in general not an immediate consequence of executing a certain action. Avoiding this behavior is one of the central aspects in our research, as it is an important prob-

lem in many domains, like for example robotics. A vacuum cleaning robot facing a descending staircase should use the safest escape plan or even not move at all. Due to the lack of space, we will not discuss learning the actions. It was shown how to learn action outcomes in planning domains, for example by the means of CPT-L (Thon et al. 2009). A straightforward mapping onto ProbLog exists. This paper also describes how the well known planning language PPDDL can be mapped to CPT-L. This implies that it is possible to generate plans for PPDDL by means of ProbLog. The idea of compiling probabilistic models described in one language into a second one has received a lot of attention in recent years. As described above this also works for the widely accepted planning language PPDDL. We believe that most other formalisms can be compiled into the language described below. This approach has the advantage that representational questions like the frame problem and composition of actions can be solved in a high level language, while the solution can be computed using the low level language.

### Probabilistic programming

A PPL consists of a set of *elementary random predicates/procedures* (ERP) and a deterministic program written in a host language. This program specifies how the joint distribution can be calculated starting from the ERPs as a generative process. While this definition covers all existing PPLs, we focus on ProbLog which is an extension of Prolog. More specifically, we will use the DTProbLog variant. Next to the ERPs, DTProbLog adds constructs to the ProbLog language that model choices or actions the agent can decide upon. We will use the term *elementary random choice* (ERC) to denote these constructs. We illustrate our setting on a simple example where an agent can choose between either throwing two coins or one thumbtack. The goal is to obtain heads up (h). The ERPs `coin` and `thumbt` are combined

```
0.5 :: coin(X, t). 0.4 :: thumbt(t).
coin(X) :- if((coin(1, t), coin(2, t)), X = t, X = h).
thumbt(h) :- not thumbt(t).
```

by stating that if both coin tosses yield tail the result will be tail, otherwise head. For planning problems we also have to specify the decisions the agent has, by giving the distribution over the available choices, which is indicated by `? :: decide(thumbt)`. The following subprogram

```
result(X) :- if(decide(thumbt), thumbt(X), coin(X)).
```

combines the choice with the distributions to form the expected result. Additionally, we take into account that throwing a thumbtack has a low probability of injuring the agent:

```
0.1 :: badluck. blind :- decide(thumbt), badluck.
```

### Finding Plans

It has been proposed (Yoon and Kambhampati 2007) to use the *Most Probable Explanation* (MPE) of the goal to find the best plan. The MPE of `result(h)` in the previous example will maximize over *all* ERPs and ERCs. This would lead to an explanation containing `thumb(t)`, `decide(thumbt)` which has an explanation probability of 0.6. This is because all explanations containing `not decide(thumbt)` have explanation probability 0.25. But on the other hand it is clearly

a wrong decision as the joint probability of a successful outcome for tossing coins is the sum of those, which is 0.75. Finding the proof with the highest explanation probability by maximizing over the ERCs and marginalizing the ERPs out is non-trivial. The corresponding algorithm is used as first step in the exact solution algorithm for DTProbLog. Instead, the solution is to find probabilities  $\theta$  that maximize the chance of reaching the goal  $P(goal|\theta)$ . These probabilities correspond to the choices the agent can make in order to reach the goal. This can be solved using a gradient descent method, developed for ProbLog as *Learning from Entailment* (LFE) (Gutmann et al. 2008), where the learner gets a set of training examples consisting of facts and the target probability. In our setting, the example is the goal with target probability 1.0, such that LFE maximizes the probability of the goal while the probabilities of the ERP are fixed in advance. This solves normal probabilistic planning. It might, however, be desirable to avoid certain proposition, such as `blind` in the previous example, or to avoid certain actions when the agent has not yet collected enough information. Therefore, we adapted the LFE setting as follows. A goal for our algorithm is a set of tuples consisting of a probability bound on a query and a weight, e.g. `{(result(head) >= 1.0, 1), (blind < 0.1, 10)}`, specifying that we want to achieve heads with a risk of injury that is below 0.1, where the latter is ten times more important than the former. This is a new problem setting – Learning from partially bounded examples or probabilistic linear programming – where each constraint specifies in which range the probability of a certain fact should be.

### Conclusion

We argued that modeling planning problems in ProbLog is straightforward but the choice of the right algorithm often non-obvious. The solution we propose extends ProbLogs Learning from Entailment setting towards learning probability values consistent with given probability bounds, a special case of the inference problem introduced by Nilsson for his probabilistic logics (Nilsson 1986).

### References

- Gutmann, B.; Kimmig, A.; De Raedt, L.; and Kersting, K. 2008. Parameter learning in probabilistic databases: A least squares approach. In *ECML PKDD*.
- Nilsson, N. J. 1986. Probabilistic logic. *AI* 28(1):71–87.
- Thon, I.; Gutmann, B.; van Otterlo, M.; Landwehr, N.; and Raedt, L. D. 2009. From non-deterministic to probabilistic planning with the help of statistical relational learning. In *ICAPS 2009 - Workshop on Planning and Learning*.
- Van den Broeck, G.; Thon, I.; Van Otterlo, M.; and De Raedt, L. 2010. DTProbLog: A decision-theoretic probabilistic Prolog. In *AAAI*.
- Yoon, S., and Kambhampati, S. 2007. Towards Model-lite Planning: A Proposal For Learning & Planning with Incomplete Domain Models. In *ICAPS*.
- Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *ICAPS*.