The Workshops of the Thirty-Second
AAAI Conference on Artificial Intelligence

Probabilistic Planning by Probabilistic Programming

Vaishak Belle
University of Edinburgh and Alan Turing Institute
vaishak@ed.ac.uk

Abstract

Automated planning is a major topic of research in artificial intelligence, and enjoys a long and distinguished history. The classical paradigm assumes a distinguished initial state, comprised of a set of facts, and is defined over a set of actions which change that state in one way or another. Planning in many real-world settings, however, is much more involved: an agent's knowledge is almost never simply a set of facts that are true, and actions that the agent intends to execute never operate the way they are supposed to. Thus, probabilistic planning attempts to incorporate stochastic models directly into the planning process. In this article, we briefly report on probabilistic planning through the lens of probabilistic programming: a programming paradigm that aims to ease the specification of structured probability distributions. In particular, we provide an overview of the features of two systems, HYPE and ALLEGRO, which emphasise different strengths of probabilistic programming that are particularly useful for complex modelling issues raised in probabilistic planning. Among other things, with these systems, one can instantiate planning problems with growing and shrinking state spaces, discrete and continuous probability distributions, and non-unique prior distributions in a first-order setting.

Introduction

Automated planning is a major topic of research in artificial intelligence, and enjoys a long and distinguished history (Fikes and Nilsson 1971). The classical paradigm assumes a distinguished initial state, comprised of a set of facts, and is defined over a set of actions which change that state in one way or another. Actions are further characterised in terms of their applicability conditions, that is, things that must be true for the agent to be able to execute it, and effects, which procedurally amounts to adding new facts to a state while removing others. The scientific agenda is then to design algorithms that synthesise a sequence of actions that takes the agent from an initial state to a desired goal state.

From the early days, automated planning was motivated by robotics applications. But it was observed that the real world – or more precisely, the robot's knowledge about the world – is almost never simply a set of facts that are true, and actions that the agent intends to execute never operate the way they are supposed to. One way to make sense of this complication is to separate the "high-level reasoning," in our case the planner's search space, from the low-level sensor-motor details. On the positive side, such a move allows the plan representation to be finite, discrete and simple. On the negative side, significant expert knowledge has to go into materialising this separation of concerns, possibly at the loss of clarity on the behaviour of the system as a whole.

Incidentally, by testing the robot's effectors repeatedly in a controlled environment, one can approximate the uncertain effects of an action in terms of a probability distribution. Similarly, based on minimalistic assumptions about the environment, expressed as a probabilistic prior, by repeated sampling, the robot can update its prior to converge on a reasonable posterior that approximates the environment (Thrun, Burgard, and Fox 2005). To that end, probabilistic planning attempts to incorporate such models directly into the planning process. There are to-date numerous languages and algorithmic frameworks for probabilistic planning, e.g., (Domshlak and Hoffmann 2007; Boutilier, Dean, and Hanks 1999; Kaelbling, Littman, and Cassandra 1998; Ong et al. 2010).

In this article, we briefly report on probabilistic planning through the lens of *probabilistic programming* (Gordon et al. 2014). Probabilistic programming is a programming paradigm that aims to ease the specification of structured probability distributions; these languages are developed so as to enable modularity and re-use in probabilistic machine learning applications. Their atomic building blocks incorporate stochastic primitives, and the formal representation also allows for compositionality. Here, we specifically provide an overview of the features of two kinds of systems, both of which have their roots in logic programming:

- HYPE (Nitti, Belle, and Raedt 2015): a planning framework based on *distributional clauses* (Gutmann et al. 2011); and
- ALLEGRO (Belle and Levesque 2015): a high-level control programming framework that extends GOLOG (Reiter 2001).

These two systems emphasise different strengths of probabilistic programming, which we think are particularly useful for complex modelling issues raised in probabilistic planning. HYPE can easily describe growing and shrinking state spaces owing to uncertainty about the existence of objects, and thus is closely related to BLOG models (Milch et al. 2007; Srivastava et al. 2014). Since HYPE is an extension of PROBLOG (Raedt, Kimmig, and Toivonen 2007), it stands to benefit from a wide range of applications and machine learning models explored with PROBLOG.[1] The dynamical aspects of the domain are instantiated by reifying time as an argument in the predicates, and so it is perhaps most appropriate for finite horizon planning problems.

ALLEGRO treats actions as first-class citizens and is built on a rich model of dynamics and subjective probabilities, which allows it to handle context-sensitive effect axioms, and non-unique probability measures placed on first-order formulas. GOLOG has also been widely used for a range of applications that apply structured knowledge (e.g., ontologies) in dynamical settings (Lakemeyer and Levesque 2007), and ALLEGRO stands to inherit these developments. GOLOG has also been shown as a way to structure search in large plan spaces (Baier, Fritz, and McIlraith 2007). Finally, since there are constructs for iteration and loops, such programs are most appropriate for modelling non-terminating behaviour (Claßen and Lakemeyer 2008).

In the sequel, we describe the essential formal and algorithmic contributions of these systems before concluding with open computational issues.

HYPE

PROBLOG aims to unify logic programming and probabilistic specifications, in the sense of providing a language to specify distributions together with the means to query about the probabilities of events. As a very simple example, to express that the object c is on the table with a certain probability, and that all objects on the table are also in the room, we would write (free variables are assumed to be quantified from the outside):

$$.6 :: onTable(c).$$
$$inRoom(x) \leftarrow onTable(x).$$

This then allows us to query the probability of atoms such $inRoom(c)$.

A more recent extension (Gutmann et al. 2011) geared for continuous distributions and other infinite event-space distributions allows the head atom of a logical rule to be drawn from a distribution directly, by means of the following syntax:

$$h \sim D \leftarrow b_1, \ldots, b_n.$$

For example, suppose there is an urn with an unknown number of balls (Milch et al. 2007). Suppose we pull a ball at a time and put it back in the urn, and repeat these steps (say) 6 times. Suppose further we have no means of identifying if the balls drawn were distinct from each other. A probabilistic program for this situation might be as follows:

[1] dtai.cs.kuleuven.be/problog

$$n \sim poisson(6).$$
$$pos(x) \sim uniform(1, 10) \leftarrow between(1, \simeq(n), x).$$

For simplicity, we assume here that the physical form of the urn is a straight line of length 10, and the position of a ball is assumed to be anywhere along this line.

HYPE is based on a dynamic extension that allows us to temporally index the truth of atoms, and so can be used to reason about actions. For example, the program:

$$numBehind(x, t+1) \sim poisson(1) \leftarrow removeObj(x, t).$$

says that on removing the object x at t, we may assume that there are objects – typically one such object – behind x. Such programs can be used in object tracking applications to reason about occluded objects (Nitti 2016).

A common declaration in many robotics applications (Thrun, Burgard, and Fox 2005) is to define actions and sensors with an error profile, such as a Gaussian noise model. These can be instantiated in HYPE using:

$$pos(x, t+1) \sim gaussian(\simeq(pos(x,t)) + 1, var)$$
$$\leftarrow move(x, t).$$
$$obs(x, t+1) \sim gaussian(\simeq(pos(x,t)), var).$$

The first rule says that the position of x on doing a move action is drawn from a normal distribution whose mean is x's current position incremented by one. The second one says that observing the current position of x is subject to additive Gaussian noise.

As an automated planning system, HYPE instantiates a Markov decision process (MDP) (Puterman 1994). Recall that MDPs are defined in terms of states, actions, stochastic transitions and reward functions, which can be realised in the above syntax using rules such as:

$$poss(act, t) \leftarrow conditions(t).$$
$$reward(num, t) \leftarrow conditions(t).$$

To compute a policy, which is a mapping from states and time points to actions, HYPE combines importance sampling and SLD resolution to effectively bridge the high-level symbolic specification and the probabilistic components of the programming model. HYPE allows states and actions to be discrete or continuous, yielding a very general planning system. Empirical evaluations are reported in (Nitti, Belle, and Raedt 2015) and (Nitti et al. 2017).

ALLEGRO

The GOLOG language has been successfully used in a wide range of applications involving control and planning (Lakemeyer and Levesque 2007), and is based on a simple ontology that all changes are a result of named actions (Reiter 2001). An initial state describes the truth values of properties, and actions may affect these values in non-trivial context-sensitive ways. In particular, GOLOG is a programming model where executing actions are the simplest instructions in the program, upon which more involved constructions for iteration and loops are defined. For example, a program to clear a table containing an unknown number of blocks would be as follows:

$$([\pi x\ onTable(x)?; removeObj(x)])^*; \neg\exists x\ onTable(x)?$$

Here, π is the non-deterministic choice of argument, semicolon denotes sequence, ? allows for test conditions, and $*$ is unbounded iteration. The program terminates successfully because the sub-program before the final test condition removes every object from the table.

As argued in (Lakemeyer and Levesque 2007), the rich syntax of GOLOG allows us, on the one hand, to represent policies and plans in an obvious fashion; for example:

$$a_1; \ldots ; a_n; P?$$

ensures that the goal P is true on executing the sequence of actions. However, the syntax also allows open-ended search; for example:

$$while\ \ \neg P\ \ \pi a.\ a$$

tries actions until P is made true. The benefit of GOLOG then is that it allows us to explore plan formulations between these two extremes, including partially specified programs that are completed by a meta-language planner.

ALLEGRO augments the underlying ontology to reason about probability distributions over state properties, and allow actions with uncertain (stochastic) effects. In logical terms, the semantical foundations rests on a rich logic of belief and actions. Consequently, it can handle partial probabilistic specifications. For example, one can say c is on the table with a certain probability as before: $pr(onTable(c)) = .6$, but it is also possible to express the probability that there is an object on the table without knowing which one: $pr(\exists x\ onTable(x)) = .6$. We can go further and simply say that there is a non-zero probability of that statement: $pr(\exists x\ onTable(x)) > 0$, which means that any distribution satisfying the formula is admissible. Such a feature can be very useful: for example, in (Kaelbling and Lozano-Pérez 2013), it is argued that when planning in highly stochastic environments, it is useful to allow a margin of error in the probability distributions defined over state properties.

To model the case of Gaussian error models, actions with uncertain effects are given a general treatment. For one thing, the effects of actions are axiomatised using the notion of successor state axioms which incorporate Reiter's solution to the frame problem (Reiter 2001). So, for example, changing the position of an object using a move action can be expressed as:

$$\begin{aligned} pos(x, do(a,s)) = u \equiv \\ (a = move(x,y) \wedge pos(x,s) = u + y) \\ \vee(a \neq move(x,y) \wedge pos(x,s) = u) \end{aligned}$$

This says that if the action of moving x was executed, its position (along a straight line) is decremented by y units, and for all other actions, the position is unaffected. To deal with uncertain effects, we will distinguish between what the agent intends and what actually happens. That is, let $move(x,y,z)$ be a new action type, where y is what the agent intends, and z is what happens. Then, the successor state axiom is rewritten as follows:

$$\begin{aligned} pos(x, do(a,s)) = u \equiv \\ (a = move(x,y,z) \wedge pos(x,s) = u + z) \\ \vee(a \neq move(x,y,z) \wedge pos(x,s) = u) \end{aligned}$$

The story remains essentially the same, except that z determines the actual position in the successor state, but it is not in control of the agent. A Gaussian error profile can be accorded to this action by means:

$$l(move(x,y,z),s) = gaussian(z; y, var)$$

That is, the actual value is drawn from a Gaussian whose mean is the intended argument y. Analogously, attributing additive Gaussian noise in a sensor for observing the position is defined using:

$$l(obs(x,z),s) = gaussian(z; pos(x,s), var)$$

That is, the observation z is drawn from a Gaussian whose mean is the actual position of the object x.

As hinted above, as an extension to GOLOG, the syntax of ALLEGRO is designed to compactly represent full or partial plans and policies in a general way, and on termination, ALLEGRO programs can be tested for any probabilistic or expectation-based criteria. The foundations of ALLEGRO was established in (Belle and Levesque 2015) with a discussion on its empirical behaviour against a predecessor based on goal regression.

Conclusions

Automated planning is often deployed in an application context, and in highly stochastic and uncertain domains, the planning model may be derived from a complex learning and reasoning pipeline, or otherwise defined over non-trivial state spaces with unknowns. In this article, we reported on two probabilistic programming systems to realise such pipelines. Indeed, combining automated planning and probabilistic programming is receiving considerable attention recently, e.g., (Srivastava et al. 2014). These languages are general purpose, and their first-order expressiveness can not only enable a compact codification of the domain but also achieve computational leverage.

One of the key concerns with the use of probabilistic programming and stochastic specifications generally is that most systems perform inference by Monte Carlo sampling. As is well-known, one is able to only obtain asymptotic guarantees with such methods, and moreover, handling low-probability observations can be challenging. In that regard, there have been recent logical approaches for inferring in mixed discrete-continuous probability spaces with tight bounds on the computed answers (Belle, Van den Broeck, and Passerini 2015; Belle, Passerini, and Van den Broeck 2015; Chistikov, Dimitrova, and Majumdar 2015). Since HYPE, ALLEGRO and many such systems use probabilistic inference as a fundamental computational backbone, the question then is whether the aforementioned approaches can enable robust planning and programming frameworks in stochastic domains.

References

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proc. ICAPS*, 26–33.

Belle, V., and Levesque, H. J. 2015. Allegro: Belief-based programming in stochastic dynamical domains. In *IJCAI*.

Belle, V.; Passerini, A.; and Van den Broeck, G. 2015. Probabilistic inference in hybrid domains by weighted model integration. In *IJCAI*.

Belle, V.; Van den Broeck, G.; and Passerini, A. 2015. Hashing-based approximate probabilistic inference in hybrid domains. In *UAI*.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11(1):94.

Chistikov, D.; Dimitrova, R.; and Majumdar, R. 2015. Approximate counting in smt and value estimation for probabilistic programs. In *TACAS*, volume 9035. 320–334.

Claßen, J., and Lakemeyer, G. 2008. A logic for non-terminating golog programs. In *KR*, 589–599.

Domshlak, C., and Hoffmann, J. 2007. Probabilistic planning via heuristic forward search and weighted model counting. *JAIR* 30:565–620.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proc. IJCAI*, 608–620.

Gordon, A. D.; Henzinger, T. A.; Nori, A. V.; and Rajamani, S. K. 2014. Probabilistic programming. In *Proc. International Conference on Software Engineering*.

Gutmann, B.; Thon, I.; Kimmig, A.; Bruynooghe, M.; and De Raedt, L. 2011. The magic of logical inference in probabilistic programming. *TPLP* 11:663–680.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *I. J. Robotic Res.* 32(9-10):1194–1227.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1–2):99 – 134.

Lakemeyer, G., and Levesque, H. J. 2007. Cognitive robotics. In *Handbook of Knowledge Representation*. Elsevier. 869–886.

Milch, B.; Marthi, B.; Russell, S.; Sontag, D.; Ong, D.; and Kolobov, A. 2007. BLOG: Probabilistic models with unknown objects. *Introduction to statistical relational learning* 373.

Nitti, D.; Belle, V.; and Raedt, L. D. 2015. Planning in discrete and continuous markov decision processes by probabilistic programming. In *ECML*.

Nitti, D.; Belle, V.; De Laet, T.; and De Raedt, L. 2017. Planning in hybrid relational mdps. *Machine Learning* 1–28.

Nitti, D. 2016. *Hybrid Probabilistic Logic Programming*. Ph.D. Dissertation, KU Leuven.

Ong, S. C. W.; Png, S. W.; Hsu, D.; and Lee, W. S. 2010. Planning under uncertainty for robotic tasks with mixed observability. *Int. J. Rob. Res.* 29(8):1053–1068.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1st edition.

Raedt, L. D.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic prolog and its application in link discovery. In *Proc. IJCAI*, 2462–2467.

Reiter, R. 2001. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT Press.

Srivastava, S.; Russell, S. J.; Ruan, P.; and Cheng, X. 2014. First-order open-universe pomdps. In *UAI*, 742–751.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic Robotics*. MIT Press.