Classical, Probabilistic, and Contingent Planning: Three Models, One Algorithm

Héctor Geffner
Departamento de Computación
Universidad Simón Bolívar
Aptdo. 89000, Caracas 1080-A
Venezuela
hector@usb.ve

Abstract

Various forms of planning in AI can be viewed as problems of sequential decision that differ only on whether the effects of actions are predictable and/or observable. Three simple mathematical models, SEARCH, MDPs and POMDPs, provide the conceptual framework to understand such problems but suitable languages and algorithms are needed to model and solve them effectively. In this paper, we analyze planning from this perspective and report the performance of a simple but general planning algorithm on various planning tasks.

Actions

We take an abstract view of planning as a problem of sequential decision in which actions have to be executed to achieve a goal. We distinguish two important aspects of actions: whether their effects are *predictable*, and whether they are *observable*. We assume throughout that time is discrete.

In *deterministic* action models, the effect of actions is completely predictable and can be represented by transition function f such that $s' = f(s, a)$ is the unique state that follows the state s after action a. *Probabilistic* action models, on the other hand, are represented by means of transition probabilities $P_a(s'|s)$ whose sum over the possible successor states s' of s and a must add up to one.

Observations provide feedback from the environment and determine the *form* of plans: in the absence of feedback, the plan is a sequence of actions; in the presence of feedback, the plan is a function of the observations. We will distinguish three cases; when the effect of actions is *fully observable*, *partially observable*, or *not observable* at all.

Plans

We refer to planning in the presence of observations as *closed-loop planning*, and planning in the absence of observations as *open-loop planning* [23, 13, 1]. Classical planning is open-loop planning, while contingent and reactive planning are forms of closed-loop planning. The two forms of planning depend on assumptions about observability and not on whether actions are deterministic or probabilistic.

Closed-loop planning is normally regarded as superior because it is more robust: it can recover from perturbations (e.g., a block falling off the gripper) and errors in the initial conditions or action models (e.g., like assuming that actions are deterministic when they are not). Open-loop plans cannot recover. For this reason, the *execution* of open-loop plans is usually extended with mechanisms for plan switching, replanning, etc.

Models

Three standard mathematical models of sequential decisions allow us to make precise what open and closed-loop plans are, and how they can be derived: SEARCH models, Markov Decision Processes or MDPs, and Partially Observable Markov Decision Processes or POMDPs. The use of MDPs in Planning has been advocated by a number of authors, e.g., [27, 7, 1, 25].

Search Models

SEARCH models [22] are characterized by three assumptions: the initial state is completely known, actions are deterministic, and effects are not observable. Formally a SEARCH problem comprises

- a state space S
- an initial state $s_0 \in S$
- actions $A(s) \subseteq A$ applicable in each state $s \in S$
- a transition function $f(s, a)$ for $s \in S$ and $a \in A(s)$
- action costs $c(a, s) > 0$
- a set $G \subseteq S$ of goal states

A *solution* of a SEARCH problem is a sequence of actions $a_0, a_1, \ldots, a_n$ that generates a state trajectory s_0, $s_1 = f(s_0)$, $\ldots$, $s_{n+1} = f(s_i, a_i)$ such that each action a_i is applicable in s_i and s_{n+1} is a goal state, i.e., $a_i \in A(s_i)$ and $s_{n+1} \in G$. The solution is *optimal* when the total cost $\sum_{i=0}^{n} c(s_i, a_i)$ is minimized.

Classical planning, i.e., open-loop planning with complete knowledge of the initial situation and deterministic actions, is a SEARCH problem where states are represented by sets of atoms, action costs are all equal, and both the transition function f and the sets of executable actions $A(s)$ are defined in a high-level language such as Strips. The *computational* problem in classical planning has been approached from a different perspective, usually based on looking at planning as a nearly decomposable problem [28]. Recent proposals suggest other formulations and algorithms that may scale up better [3, 17]. In [5] we suggest that SEARCH methods can scale up well too provided a suitable 'planning' heuristic function is used (see also [21]).

MDPs

Markov Decision Processes (MDPs) [24, 2] differ from SEARCH in two main respects: they accommodate *probabilistic actions*, and they assume that the effect of actions is *fully observable*. An MDP is thus given by:[1]

- a state space S
- actions $A(s) \subseteq A$ applicable in each state $s \in S$
- transition probabilities $P_a(s'|s)$ for $s \in A$ and $a \in A(s)$
- action costs $c(a, s) > 0$
- a set $G \subseteq S$ of goal states

The state s_{i+1} that results from a state s_i and an action a_i is *not predictable* but is *observable*, and hence provides feedback for selecting the next action a_{i+1}. As a result, a *solution* of an MDP is not an action sequence, but a function π mapping states s into applicable actions $a \in A(s)$. Such a function is called a *policy.* A policy π assigns a probability to every state trajectory $s_0, s_1, s_2, \ldots$ starting in a state s_0, that is given by the product of all transition probabilities $P_{a_i}(s_{i+1}|s_i)$ where $a_i = \pi(s_i)$. We assume that actions in goal states have no costs and no effects (i.e., $c(a, s) = 0$ and $P_a(s|s) = 1$ if $s \in G$). The *expected cost* associated with a policy π starting in state s is the weighted average of the probability of such trajectories times their cost $\sum_{i=0}^{\infty} c(\pi(s_i), s_i)$. An *optimal solution* is a control policy π^* that has a minimum expected cost for all states $s \in S$.

While Classical Planning can be formulated as a SEARCH problem, Closed-loop Planning with *complete information* can be formulated as an MDP [11]. The desired closed-loop plans are the optimal policies π^*.

POMDPs

POMDPs generalize MDPs allowing the state to be *partially observable* [26, 8]. Information about the state comes from observations o whose probabilities $P_a(o|s)$ depend on the action a performed and the unobserved but true resulting state s. In addition, a prior probability distribution over the states encodes the prior belief about the initial state of the environment. A POMDP is thus characterized by:

- states $s \in S$
- actions $A(s) \subseteq A$ applicable in each state s
- costs $c(a, s) > 0$ of performing action a in s
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- initial belief state b_0
- final belief states b_F
- observations o after action a with probabilities $P_a(o|s)$

Since feedback from the environment is only partial, the *solution* of a POMDP is not a function mapping states into actions, but a function mapping *belief states* into actions, where belief states b are probability distributions over the real states s of the environment. The effect of the actions on *belief states* is completely predictable. Indeed the belief state b_a that results from performing action a in the belief state b is:

$$b_a(s) = \sum_{s' \in S} P_a(s|s') b(s') \tag{1}$$

while the belief state b_a^o that results from performing action a in b and then observing o is

$$b_a^o(s) = P_a(o|s) b_a(s) / b_a(o) \tag{2}$$

where $b_a(o)$ is the probability of observing o after doing a in b given by

$$b_a(o) = \sum_{s \in S} P_a(o|s) b_a(s) \tag{3}$$

The effect of actions and observations is thus to transform a belief state b into a new belief b_a^o with probability $b_a(o)$. The planning task is to go from an initial belief state b_0 to a final belief state b_F at a minimum expected cost. This is nothing else but an MDP over *belief space,* where the real states s of the environment have been replaced by belief states b. The 'goal' belief states b_F can be defined in various ways; as the belief states in which we are certain to have reached a real goal state (i.e., $b(s) = 0$ for $s \notin G$), as the belief states in which we are pretty sure to have reached a goal state (i.e., $\sum_{s \in G} b(s) > 1 - \epsilon$), etc.

Closed-loop Planning with Incomplete Information is a POMDP problem whose solutions are the closed-loop policies mapping belief states into actions. In the AI literature, such policies are often represented by contingent plans, i.e., sequential plans extended with tests and branches [14, 10, 20]. All POMDP policies can be represented in this way.

Computation

Rather than providing a review of algorithms for solving MDPs and POMDPs, we focus on a simple algorithm, called RTDP, that can effectively solve complex instances of both, when good estimates of the expected costs are available and the initial (belief) state is known a priori. RTDP (Real Time Dynamic Programming) is

[1] We are considering a subclass of MDPs, the so-called *stochastic shortest-path* MDPs [2]. For general treatments, see [2] and [24].

a generalization of Korf's LRTA* [18] due to [1], that combines ideas from real-time search and dynamic programming. RTDP is also closely related to the algorithms used in Reinforcement Learning [1, 2, 27].

RTDP is basically a *greedy* or *hill-climbing* algorithm that from any state s searches for a goal state using estimates $V(s)$ of the *expected cost* to reach the goal. The main difference with standard hill-climbing is that these estimates are updated dynamically. Initially $V(s)$ is set to $h(s)$, where h is a suitable *heuristic function*, and every time an action a from s is taken, $V(s)$ is updated to make it consistent with the estimates of its successor states (Step 3, Fig. 1). The updates guarantee two things: first, that in any *single trial* the algorithm will eventually find the goal, and second, that after *repeated trials*, the cost estimates will eventually converge to their optimal values, provided that the heuristic function h used to initialize the estimates is *admissible* (non-overestimating) [1, 18].

1 **Evaluate** each action a applicable in s as:

$$Q(a, s) = c(a, s) + \sum_{s' \in S} P_a(s'|s)V(s')$$

initializing $V(s')$ to $h(s')$ when needed

2 **Apply** action $\mathbf{a}$ with minimum $Q(\mathbf{a}, s)$ value, breaking ties randomly

3 **Update** $V(s)$ to $Q(\mathbf{a}, s)$

4 **Observe** resulting state s'

5 **Exit** if s' is a goal, else set s to s' and go to 1

Figure 1: RTDP Loop

In the implementation, the estimates $V(s)$ are stored in a hash table that initially contains an estimate $V(s_0)$ for the initial state s_0 only. Then when the value $V(s')$ of a state s' that is not in the table is needed, a new entry with $V(s')$ set to $h(s')$ is created.

RTDP is an 'anytime' algorithm for solving MDPs in the sense that after any number of steps and trials, the hash table along with the heuristic function determines a greedy control policy that eventually converges to the optimal policy when h is admissible. Since POMDPs correspond to MDPs over belief states, RTDP can be used to solve POMDPs as well by simply substituting the current state s in the RTDP loop in Fig. 1 by the current *belief state* b, and the observed state s' by the new belief state b_a^o (Equations 1–3), where o is the observation obtained. Problems of *open-loop* planning can be solved in a similar way, as they are a special case of problems of closed-loop planning with incomplete information in which are *no* observations, and hence $b_a^o = b_a$. This includes, for example, the problems considered by the probabilistic planner BURIDAN [19].

Representation

SEARCH, MDP and POMDP models are useful for analysis but not for modeling. In AI it has been a standard practice to model planning problems by means of high-level languages such as Strips [15]. In recent years similar languages have been defined for modeling probabilistic actions [19, 12] and general POMDP models [16]. We illustrate the latter with a problem of planning with incomplete information due to Levesque [20]. It involves an agent that has a large supply of eggs and whose goal is to get three good eggs and no bad ones into one of two bowls. The eggs can be either good or bad, and at any time the agent can find out whether a bowl contains a bad egg by inspecting the bowl. In [16] this problem is encoded by expressions such as the ones below, which are automatically compiled into a POMDP whose solution, obtained by the RTDP algorithm, provides the desired plan.

Action:	**grab-egg**()
Precond:	$\neg holding$
Effects:	$holding :=$ **true**
	$good? :=$ (**true** 0.5 ; **false** 0.5)
Action:	**break-egg**$(bowl : BOWL)$
Precond:	$holding \wedge (ngood(bowl) + nbad(bowl)) < 4$
Effects:	$holding :=$ **false**
	$good? \rightarrow ngood(bowl) := ngood(bowl) + 1$
	$\neg good? \rightarrow nbad(bowl) := nbad(bowl) + 1$
Action:	**pour**$(b1 : BOWL, b2 : BOWL)$
Precond:	$(b1 \neq b2) \wedge \neg holding$
	$ngood(b1) + nbad(b1) + ngood(b2) + nbad(b2) < 4$
Effects:	$ngood(b1) := 0 \;,\; nbad(b1) := 0$
	$ngood(b2) := ngood(b2) + ngood(b1)$
	$nbad(b2) := nbad(b2) + nbad(b1)$
Action:	**clean**(bowl:BOWL)
Precond:	$\neg holding$
Effects:	$ngood(bowl) := 0 \;,\; nbad(bowl) := 0$
Action:	**inspect**$(bowl : BOWL)$
Effect:	**obs**$(nbad(bowl) > 0)$

The language extends Strips in several ways: states are not associated with sets of atoms but with assignments to arbitrary fluents; probabilities, costs and primitive operations like '+' are included, and a special predicate **obs** is used to indicate observability. The fluents in this problem are the number of good eggs and bad eggs in each bowl ($ngood(a)$, $ngood(b)$, $nbad(a)$, $nbad(b)$), and the boolean variables *holding*? and *good*? that represent whether the agent is currently holding an egg and whether such an egg is good. The fluent *holding* is always observable, but the value of the expression $nbad(nbowl) > 0)$ is only observable after doing the action *inspect*. For the formal syntax and semantics of this language, see [16].

Results

Classical Planning

Tables 2 and 3 show results comparing RTDP with two recent and powerful planners, GRAPHPLAN [3] and SATPLAN [17], over the suite of problems in [17]. These tables are from [6], except that the RTDP column shows

Problem	GRAPHPLAN	SATPLAN	RTDP
rocket_ext.a	268	0.17	1.3
logistics.a	5,942	22	7
logistics.b	2,538	6	6
12 blocks	1,119	18	1
15 blocks	—	524	5
19 blocks	—	4,220	19

Figure 2: Time performance in seconds of RTDP in comparison with GRAPHPLAN and SATPLAN

Problem	GRAPHPLAN	SATPLAN	RTDP
rocket_ext.a	34	34	35/28
logistics.a	54	54	64/57
logistics.b	47	47	58/48
12 blocks	9	9	16/12
15 blocks	—	14	24/19
19 blocks	—	18	32/25

Figure 3: Quality performance. Averages and minimal plan lengths over 25 runs shown

the result of the algorithm with no lookahead.[2] The heuristic used is very informative but is *not* admissible [6]. For this reason the algorithm does not improve much after successive trials, and hence, only a single trial is considered.

As it can be seen from the tables, RTDP reaches the goal very fast (Fig. 1), but the length of plans is sometimes far from optimal (Fig. 2). One way to decrease the average length of plans is thus to run the algorithm several times keeping only the best run. Other methods are the addition of 'noise' in the selection of actions (see [6]), increased lookahead, and variations on the heuristic function. In principle RTDP can produce optimal plans, but cannot guarantee the optimality of the plans produced as planners such as GRAPHPLAN.

Planning with Incomplete Information

Figures 4 displays the performance curve for the Omelette problem discussed above. The curve that is flat shows the average number of actions to solve the problem for the obvious plan in which an egg is grabbed and broken it into one of the two bowls, and after inspecting the bowl, it's either passed to the other bowl or discarded, until three eggs have been passed. The other curve shows the performance of the greedy controller produced by RTDP after different number of trials. The heuristic function used in this case is admissible and follows from assuming that the next state will be observable [9, 16]. The convergence takes more than 1000 trials, as the algorithm has to 'learn' the value of the action 'inspect', which as all information-

[2]The algorithm in [6] is referred to as ASP for Action Selection for Planning, and is presented as a variation of Korf's LRTA* [18], which is the deterministic version of RTDP.

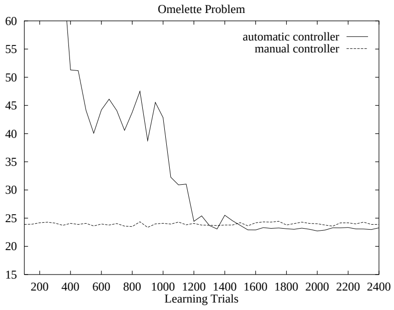

Figure 4: Comparison of RTDP policy vs. handcrafted plan for the 'Omelette' problem

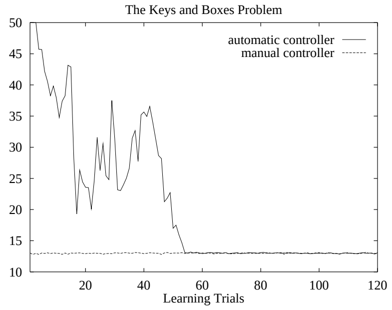

Figure 5: Comparison of RTDP policy vs. handcrafted plan for 'Key and Boxes' problem

gathering actions, appears useless to the heuristic. The time for 2000 trials in this problem is in the order of 192 seconds on an UltraSparc running at 143Mhz.

The second problem is originally from [10] where it is presented as a challenging problem for contingent planning. It deals with a robot that is inside a room where there is a table, two boxes, a pile of red things by the door, and a key that may be in either of the two boxes. The goal is to get one red thing outside the room. In order to get out of the room the key must be placed by the door. The robot cannot hold two things at the same type, and while it knows whether it's holding something or not, it does not know *what* it is holding. The resulting POMDP for this problem involves 480 states, 21 actions, and 6 observations (see [5] for a complete high-level description). An optimal plan for this problem is "go to the door, pickup a red thing, leave it on the table, then go for the key in one of the boxes and place it by the door; finally, go to the table, pick up what's in there and go outside". Other plans are possible but most plans are no good as they leave the robot in a state of knowledge from which the goal can't be achieved. The average cost of this plan is shown by the flat curve in Fig. 5. The other curve shows the performance of the RTDP controller as a function of the number of trials. The average time to compute the first 60 trials is 5.61 seconds. No times for the solution are reported in [10].

Conclusions

SEARCH, MDPs and POMDPs are useful models for understanding various forms of planning. Effective planning, however, also requires suitable languages and algorithms. From this perspective, the planning languages are a convenient way for defining these models and revealing their structure so that it can be used by suitably defined heuristics. We have also presented a simple and general algorithm that can be used for open and closed-loop planning, with either complete or incomplete information.

Approaches to planning should be judged by their facilities for modeling and solving problems, and hence a rich collection of problems would be highly valuable. Recently we have applied some of these ideas to two interesting sequential decision problems: sorting an array of numbers of fixed sized, and learning decision trees from data. Small versions of both problems are formulated and solved as problems of planning with incomplete information in [4]. The application of this method to larger versions of these problems presents a number of interesting challenges.

Acknowledgement. My work on planning and MDPs has been in collaboration with Blai Bonet. Partial support is due to Conicit, Grant S1-96001365.

References

[1] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[2] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[3] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, Montreal, Canada, 1995.

[4] B. Bonet and H. Geffner. Learning sorting and decision trees with POMDPs. To appear in *Proceedings ICML-98*, 1998.

[5] B. Bonet and H. Geffner. Planning and control with incomplete information using POMDPs: Experimental results. Available at `http://www.ldc.usb.ve/~hector`, 1998.

[6] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*. MIT Press, 1997.

[7] C. Boutilier, T. Dean, and S. Hanks. Planning under uncertainty: structural assumptions and computational leverage. In *Proceedings of EWSP-95*, 1995.

[8] A. Cassandra, L. Kaebling, and M. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings AAAI94*, pages 1023–1028, 1994.

[9] A. Cassandra, L. Kaebling, and M. Littman. Learning policies for partially observable environments: Scaling up. In *Proc. ICML-95*.

[10] G. Collins and L. Pryor. Planning under uncertainty: Some key issues. In *Proceedings IJCAI95*, 1995.

[11] T. Dean, L. Kaebling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In *Proceedings AAAI93*, pages 574–579. MIT Press, 1993.

[12] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[13] T. Dean and M. Wellman. *Planning and Control*. Morgan Kaufmann, 1991.

[14] O. Etzioni, S. Hanks, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. In *Proceedings KR'92*.

[15] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971.

[16] H. Geffner and B. Bonet. High-level plannnig and control with incomplete information using POMDP's. Available at `http://www.ldc.usb.ve/~hector`, 1998.

[17] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, pages 1194–1201, 1996. MIT Press.

[18] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[19] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.

[20] H. Levesque. What is planning in the presence of sensing. In *Proceedings AAAI-96*, pages 1139–1146, 1996. MIT Press.

[21] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. AIPS-96*.

[22] A. Newell and H. Simon. *Human Problem Solving*. Prentice–Hall, Englewood Cliffs, NJ, 1972.

[23] L. Padulo and M. Arbib. *System Theory*. Hemisphere Publishing Co., 1974.

[24] M. Puterman. *Markov Decision Processes Programming*. John Wiley and Sons, Inc., 1994.

[25] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.

[26] E. Sondik. *The Optimal Control of Partially Observable Markov Processes*. Stanford, 1971.

[27] R. Sutton. Integrated architectures for learning, planning and reacting. In *Proc. ICML-90*.

[28] D. Weld. An introduction to least commitment planning. *AI Magazine*, 1994.