Modelling Intelligent Behaviour: The Markov Decision Process Approach

Héctor Geffner

Depto de Computación
Universidad Simón Bolívar
Aptdo. 89000, Caracas, Venezuela
http://www.ldc.usb.ve/~hector

Abstract. The problem of selecting action in environments that are dynamic and not completely predictable or observable is a central problem in intelligent behavior. From an AI point of view, the problem is to design a mechanism that can select the best actions given information provided by sensors and a suitable model of the actions and goals. We call this the problem of *Planning* as it is a direct generalization of the problem considered in Planning research where feedback is absent and the effect of actions is assumed to be predictable. In this paper we present an approach to Planning that combines ideas and methods from Operations Research and Artificial Intelligence. Basically Planning problems are described in *high-level action languages* that are compiled into general mathematical models of sequential decisions known as *Markov Decision Processes* or *Partially Observable Markov Decision Processes*, which are then solved by suitable *Heuristic Search Algorithms.* The result are *controllers* that map sequences of observations into actions, and which, under certain conditions can be shown to be optimal. We show how this approach applies to a number of concrete problems and discuss its relation to work in Reinforcement Learning.

1 Introduction

The problem of selecting actions in environments that are dynamic and not completely predictable is a central problem in AI. Given a model of the actions and goals, the problem is to produce a *controller* that can map sequences of observations into suitable actions [26]. We call this the problem of *Planning* as it is a general version of problem traditionally considered in Planning research where feedback is absent and actions are assumed to be deterministic (e.g., [22, 26]). In this paper we will be concerned with this problem and show how it can be addressed by a suitable combination of *models, languages* and *algorithms.* Models will allow us to understand the problem, while languages and algorithms will allow us to represent and solve specific problem instances.

A simple planning problem is Levesque's Omelette Problem [20] that 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. A sensible plan for this problem is to follow a loop in which an egg is grabbed from the pile and broken into a bowl. If the egg is good, it's passed to the other bowl, else it is discarded. The loop is continued until three eggs have been passed.

The Planning problem is how to model problems of this type and how to obtain the corresponding plans from a suitable description of the actions and goals. As this example illustrates, the *form* of plans is intimately related to the presence or absence of *feedback*. In general, the sequence of actions depends on the observations. 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, 14, 1]. Classical planning is open-loop planning, while contingent and reactive planning are forms of closed-loop planning. While open-loop planning has been the focus of most research in AI Planning, closed-loop planning is normally regarded as superior as it is more robust. Closed-loop plans 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), while open-loop plans cannot. Closed-loop planning, however, requires more sophisticated models, languages and algorithms. Models have to make precise what closed-loop plans are, languages have to make room for both physical and information gathering actions, and algorithms have to produce *functions* mapping observation sequences into actions.

In this paper we present an approach to closed-loop planning that is based on these three elements. First we review, SEARCH, MDP, and POMDP *models* for planning, then we consider a suitable combination of heuristic search and dynamic programming *algorithms* for solving these models, and finally, we consider high-level action *languages* for representing MDPs and POMDPs in a convenient way. We have actually implemented a shell that supports this approach, and given a high-level representation of actions and goals, produces the appropriate controllers [4]. We report empirical results for a number of problems, and discuss how this approach relates to current ideas in Reinforcement Learning [30].

2 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 from suitable descriptions of actions and goals. They are SEARCH models, Markov Decision Processes or MDPs, and Partially Observable Markov Decision Processes or POMDPs.

2.1 Search Models

SEARCH models [21] are the most basic action models in AI and are characterized by three assumptions: the initial state is completely known, actions are deterministic, and their effects are not observable. Formally a SEARCH problem is comprised of

- 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 by looking at planning as a nearly decomposable problem [32]. Recent work suggests other formulations and algorithms that may scale up better [3, 17], and in [6] we argue that SEARCH methods can scale up well too, provided a suitable heuristic function is obtained from the Strips representation of the problem (see also [?]).

2.2 MDPs

Markov Decision Processes (MDPs) [25, 2] differ from SEARCH models 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

[1] We are considering a subclass of MDPs, the so-called *stochastic shortest-path* MDPs [2]. For general treatments, see [2] and [25]. For uses of MDPs in Planning and AI, see [29, 8, 1, 26].

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 [12, 1]. The desired closed-loop plans are the optimal policies π^*.

2.3 POMDPs

POMDPs generalize MDPs allowing the state to be *partially observable* [27, 9]. 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

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

Actions a thus transform a belief state b into a new belief b_a^o with probability $b_a(o)$. The planning task is to go from the 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 [15, 11, 20].

3 Algorithms

We have seen that Planning problems can be formulated as either SEARCH, MDP, or POMDP problems according to the type of feedback (observations) available to the planner at run time. Techniques for solving SEARCH problems are reviewed in details in most AI texts; e.g., [26], and include algorithms such as A* and greedy search, all based on an heuristic function $h(s)$ that estimates the cost from any state s to a goal state. A* is guaranteed to find the optimal action sequence when the heuristic function is *admissible* (i.e., does not overestimate the true cost the goal), yet it may take exponential space. Greedy search, on the other hand, takes constant space but is not guaranteed to find optimal solutions, or any solutions at all. An heuristic search algorithm for classical planning problems was shown to be competitive with the state of the art planning algorithms such as GRAPHPLAN and SATPLAN in [7]. The algorithm is basically a version of A* with a limited buffer size, and uses an heuristic function extracted from the Strips encoding of the problem to guide the search.

While extensions of the heuristic search methods such as AO* [22, 24] could be used to solve problems with *probabilistic actions*, the standard approach to solve MDPs and POMDPs is by means of *dynamic programming* methods [25, 2]. The idea is to compute from any state s (or belief state b) the optimal expected cost $V(s)$ ($V(b)$) to the reach a goal and then use these values to select the optimal actions. These optimal expected costs $V(\cdot)$ obey the following fixed point equations in SEARCH, MDPs, and POMDPs models

$$\text{SEARCH:} \quad V(s) = \min_{a \in A(s)} [c(a,s) + V(s_a)] \tag{4}$$

$$\text{MDP:} \quad V(s) = \min_{a \in A(s)} [c(a,s) + \sum_{s'} P_a(s'|s) V(s')] \tag{5}$$

$$\text{POMDP:} \quad V(b) = \min_{b \in A(b)} [c(a,b) + \sum_{o} b_a(o) V(b_a^o)] \tag{6}$$

Value iteration finds the solution to these equations by an iteration method in which initial estimates V_0 are plugged on the right hand side of the equations to yield new estimates V_1 on the left hand side, which are used again to get estimates V_2 and so on. Under suitable conditions it can be proved that the estimates V_i approach the optimal values when $i \to \infty$ [25, 2]. The operation that yields new estimates V_{i+1} in terms of old estimates V_i is called an *update*.

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

Fig. 1. RTDP Loop

Thus, each step of value iteration involves a parallel update of the estimates $V_i(s)$ for all $s \in S$.

Problems involving up to hundred of thousands of states can be solved by these methods in a reasonable amount time and memory. Yet larger problems are not uncommon in AI, and certainly arise in POMDPs where the set of belief states is continuous and infinite. The problem with methods such as value iteration is that they compute the optimal values of *all* (belief) states. In many problems, however, only the value of a small number of states matter. This occurs in particular in problems in which the initial state s (or b) is known a priori. In such cases, most of the updates in value iteration are wasted on irrelevant states.

The idea of the so-called *Real Time Dynamic Programming* (RTDP) methods [1] is to allow the solution of much larger problems by focusing the updates on the states that are likely to be relevant. This is achieved by using the current estimates $V(s)$ (or $V(b)$ in POMDPs) to guide a greedy search while limiting the updates to the states that are visited. Interestingly, repeated trials of this greedy search algorithm with updates, eventually delivers an optimal policy, *even if a large fraction of the states is never or seldom visited* [18, 1, 2]. For this reason RTDP algorithms can scale up to much larger problems provided good initial estimates are available.

The RTDP algorithm for SEARCH and MDPs is shown in Fig. 1. Note that 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 in Fig. 1, Equation 5). The updates, guarantee that in any *single trial* the algorithm will eventually find the goal, and 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 [1, 18].

In the implementation, the estimates $V(s)$ are stored in a hash table that initially contains an estimate $V(s_0)$ for 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 there are *no* observations, and hence $b_a^o = b_a$. This includes, for example, the problems considered by the probabilistic planner BURIDAN [19].

4 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 [16]. In recent years similar languages have been defined for modeling probabilistic actions [19, 13] and general POMDPs [4]. 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 [4] this problem is encoded by expressions such as the ones in Fig. 2, which are automatically compiled into a POMDP, whose solution obtained by the RTDP algorithm provides the desired plan.

Such a 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 [4].

5 Results

5.1 Classical Planning

Tables 3 and 4 show results comparing RTDP with two recent and powerful planners, GRAPHPLAN [3] and SATPLAN [17], over the suite of problems in [17].

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

Fig. 2. High-level encoding of Omelette Problem

These tables are from [7], except that the RTDP column shows the result of the algorithm with no lookahead.[2] The heuristic used is very informative but is *not* admissible [7]. 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 [7]), 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.

5.2 Planning with Incomplete Information

Figures 5 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

[2] The algorithm in [7] 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.

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

Fig. 3. 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

Fig. 4. Quality performance. Averages and minimal plan lengths over 25 runs shown

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 [10, 4]. The convergence takes more than 1000 trials, as the algorithm has to 'learn' the value of the action 'inspect', which as all information-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 [11] 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 [6] 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. 6. 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 [11].

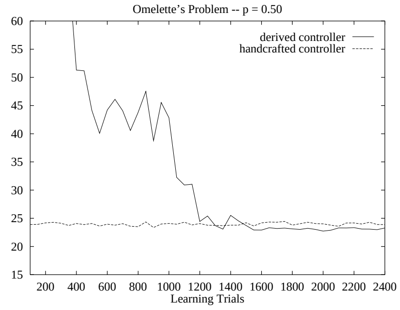

Fig. 5. Comparison of RTDP policy vs. handcrafted plan for the 'Omelette' problem

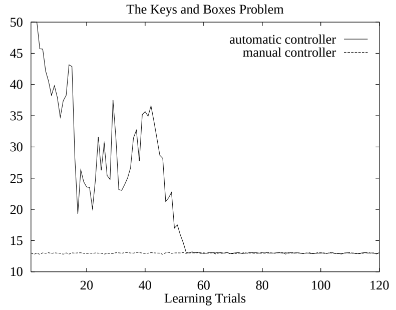

Fig. 6. Comparison of RTDP policy vs. handcrafted plan for 'Key and Boxes' problem

6 Discussion

SEARCH, MDPs and POMDPs are key models for understanding various forms of open and closed-loop 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 considered a simple and general RTDP algorithm that can be used for open and closed-loop planning with either complete or incomplete information, and presented some empirical results.

MDP models and RTDP algorithms are closely related to the models and algorithms used in Reinforcement Learning [28]. Indeed, reinforcement learning algorithms are currently understood as algorithms for solving MDPs by trial and error with no prior knowledge of cost and probabilities [31, 1]. The Q-learning algorithm in particular [31] is basically the model-free version of RTDP. For two recent books on MDPs and Reinforcement Learning; see [30] and [2].

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. High-level plannnig and control with incomplete information using POMDP's. In *Proceedings AAAI Fall Symp. on Cognitive Robotics*, 1998.
5. B. Bonet and H. Geffner. Learning sorting and decision trees with POMDPs. In *Proceedings ICML-98*, 1998.
6. B. Bonet and H. Geffner. Planning and control with incomplete information using POMDPs: Experimental results. Available at `http://www.ldc.usb.ve/~hector`, 1998.
7. B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, pages 714–719. MIT Press, 1997.
8. C. Boutilier, T. Dean, and S. Hanks. Planning under uncertainty: structural assumptions and computational leverage. In *Proceedings of EWSP-95*, 1995.
9. A. Cassandra, L. Kaebling, and M. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings AAAI94*, pages 1023–1028, 1994.
10. A. Cassandra, L. Kaebling, and M. Littman. Learning policies for partially observable environments: Scaling up. In *Proc. of the 12th Int. Conf. on Machine Learning*, 1995.
11. G. Collins and L. Pryor. Planning under uncertainty: Some key issues. In *Proceedings IJCAI95*, 1995.
12. T. Dean, L. Kaebling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In *Proceedings AAAI93*, pages 574–579. MIT Press, 1993.
13. T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.
14. T. Dean and M. Wellman. *Planning and Control.* Morgan Kaufmann, 1991.
15. O. Etzioni, S. Hanks, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. In *Proceedings of the Third Int. Conference on Principles of Knowledge Representation and Reasoning*, pages 115–125. Morgan Kaufmann, 1992.
16. R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971.
17. H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, pages 1194–1201, Protland, Oregon, 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, Portland, Oregon, 1996. MIT Press.
21. A. Newell and H. Simon. *Human Problem Solving.* Prentice–Hall, Englewood Cliffs, NJ, 1972.
22. N. Nilsson. *Principles of Artificial Intelligence.* Tioga, 1980.
23. L. Padulo and M. Arbib. *System Theory.* Hemisphere Publishing Co., 1974.

24. J. Pearl. *Heuristics.* Morgan Kaufmann, 1983.
25. M. Puterman. *Markov Decision Processes – Discrete Stochastic Dynamic Programming.* John Wiley and Sons, Inc., 1994.
26. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1994.
27. E. Sondik. *The Optimal Control of Partially Observable Markov Processes.* PhD thesis, Stanford University, 1971.
28. R. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.
29. R. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of ML-90*, pages 216–224. Morgan Kaufmann, 1990.
30. R. Sutton and A. Barto. *Introduction to Reinforcement Learning.* MIT Press, 1998.
31. C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, 1989.
32. D. Weld. An introduction to least commitment planning. *AI Magazine*, 1994.