Learning to Plan Probabilistically

Ron Sun
NEC Research Institute
and University of Alabama
4 Independence Way, Princeton, NJ
rsun@cs.ua.edu

Chad Sessions
The University of Alabama
Tuscaloosa, AL

Abstract

This paper discusses the learning of probabilistic planning without a priori domain-specific knowledge. Different from existing reinforcement learning algorithms that generate only reactive policies and existing probabilistic planning algorithms that requires a substantial amount of a priori knowledge in order to plan, we devise a two-stage bottom-up learning-to-plan process, in which first reinforcement learning/dynamic programming is applied, without the use of a priori domain-specific knowledge, to acquire a reactive policy and then explicit plans are extracted from the learned reactive policy. Plan extraction is based on a beam search algorithm that performs temporal projection in a restricted fashion guided by the value functions resulting from reinforcement learning/dynamic programming.

Introduction

Some recent AI planning systems, in a broad sense, are able to learn to plan in an uncertain (probabilistic) world, for example, Kushmerick, Hanks and Weld (1995), and Dearden and Boutilier (1996). The problem with these existing approaches is the amount of *prior* domain-specific knowledge necessary in order to make planning possible. Such a requirement begs the question of how one obtains a complete set of planning rules to begin with. On the other hand, dynamic programming (DP) or its reinforcement learning (RL) variants (Bertsekas and Tsitsiklis 1996, Barto et al 1995, Kaelbling et al 1996) may start with little or no a priori knowledge, but they only learn closed-loop policies, or reactive policies (Agre and Chapman 1989, Schoppers 1987); that is, actions are selected, in accordance with a learned policy, on a moment-to-moment basis from the current state/input, without (explicit) regard to future gains or losses. The result is not an explicit plan that can be used in an open-loop fashion (Barto et al 1995). However, explicit plans are what traditional AI planning research strives for (Sacerdoti 1974, Warren 1976, Pollack 1992) and thus important in that regard. In addition, explicit plans can also be more useful than reactive policies in many situations: E.g., An explicit plan sequence can be executed without environmental feedback, which can be very useful when feedback is not available or unreliable.

In order to learn explicit plans without a priori domain-specific knowledge, we take a bottom-up approach. That is, we first learn a simple open-loop, reactive policy (e.g. in neural networks) (Agre and Chapman 1989) through utilizing DP/RL (Bellman 1957, Bertsekas and Tsitsiklis 1996, Barto et al 1995, Watkins 1989), without the use of a priori knowledge. We also simultaneously learn a world model (cf. Sutton 1990). Then we extract an explicit plan from this learned reactive policy, based on following the most probable path(s), with probabilities being calculated from the results of the DP/RL process and the world model. In other words, our approach is learning open-loop control (traditional AI plans) from first learning closed-loop control (i.e., reactive policies).

Learning to plan

The Foundation

To implement such planning, we can use Q-learning (Watkins 1989), in which the updating is done completely on-line and incremental, without explicitly using transition probability:

$$Q(s_t, u_t) := (1-\alpha)Q(s_t, u_t)$$
$$+\alpha(g(s_t, u_t, s_{t+1}) + \gamma \max_{u_{t+1}}(Q(s_{t+1}, u_{t+1})))$$

where $\alpha \in (0,1)$ is the learning rate, u_t is determined by an action policy, e.g., $u_t = argmax_u Q(s_t, u)$ or $prob(u_t) = \frac{e^{Q(s_t,u_t)/\tau}}{\sum_u e^{Q(s_t,u)/\tau}}$, and γ is 1 for nondiscounted Q-learning. (Note that rewards, g, are always zero before the last step.) The learning can be carried out in a backpropagation neural network. We can show that Q values along the way of a path to the goal can have specific interpretations that can readily be used in probabilistic planning: The Q values are actually probabilities that indicate the likelihood of reaching the goal by performing the corresponding action and following the policy thereon.

Theorem 1 *In nondiscounted Q-learning, the optimal Q values represent the probability of reaching the goal*

states, that is,

$$Q(s,u) = \sum_i prob_i^{n_i}(s,u)$$

and

$$prob_i^{n_i}(s,u) = \Pi_{t=1}^{n_i} p_{s_{it},s_{it+1}}(u_{it})$$

where $prob_i^{n_i}(s,u)$ is the probability of path i reaching the goal, starting from state s and action u and subsequently selecting actions according to a policy P. $p_{s_{it},s_{it+1}}(u_{it})$ is the probability of the t−th state transition along the way of path i (from s_{it} to s_{it+1} with action u_{it}), $s_{i1} = s$ and $u_{i1} = u$; n_i is the length of path i and can either be finite or infinite.

We can extend the above framework to deal with discounted Q-learning too (Sun and Sessions 1998).

Extracting plans

What we need now is a method that turns a set of Q values and the corresponding policy P resulting from these values into a plan L that is in the form of a sequence of steps in accordance with the traditional AI formulation of planning. The basic idea is that we use beam search, to find the best action sequences (or conditional action sequences) that achieve the goal with a certain probability (based on the earlier theorems).

We employ the following data structures. The *current state set, CSS*, consists of multiple pairs in the form of $(s, p(s))$, in which the first item indicates a state s and the second item $p(s)$ indicates the probability of that state. For each state in CSS, we find the corresponding best action. In so doing, we have to limit the number of branches at each step, for the sake of time efficiency of the algorithm as well as the representational efficiency of the resulting plan. The set thus contains up to (a fixed number) n pairs, where n is the branching factor in beam search. A conditional plan with limited branches (widths) may fail if at a particular step, none of the listed states is present. The remedy is to include a default action for each step in the plan that is irrespective of the state at a step. In order to calculate the best default action at each step, we include a second set of states CSS', which covers a certain number (m) of possible states not covered by CSS.

PlanExtraction:

Set the current state set $CSS = \{(s_0, 1)\}$ and $CSS' = \{\}$

Repeat until the termination conditions are satisfied (e.g., $step > D$)

- For each action u, compute the probabilities of transitioning to each of all the possible next states (for all $s' \in S$) from each of the current states ($s \in CSS$):

$$p(s', s, u) = p(s) * p_{s,s'}(u)$$

- For each action u, compute its estimated utility with respect to each state in CSS:

$$Ut(s,u) = \sum_{s'} p(s', s, u) * \max_v Q(s', v)$$

That is, we calculate the *probabilities* of reaching the goal after performing action u from the current state s.

- For each action u, compute the estimated utility with respect to *all* the states in CSS':

$$Ut(CSS', u) = \sum_{s \in CSS'} \sum_{s'} p(s) * p_{s,s'}(u) \max_v Q(s', v)$$

- For each state s in CSS, choose the action u_s with the highest utility $Ut(s,u)$:

$$u_s = argmax_u Ut(s,u)$$

- Choose the best default action u with regard to all the states in CSS':

$$u = argmax_{u'} Ut(CSS', u')$$

- Update CSS to contain n states that have the highest n probabilities, i.e., with the highest $p(s')$'s:

$$p(s') = \sum_{s \in CSS} p(s', s, u_s)$$

where u_s is the action chosen for state s.

- Update CSS' to contain m states that have the highest m probabilities calculated as follows, among those states that are not in the new (updated) CSS:

$$p(s') = \sum_{s \in CSS \cup CSS'} p(s', s, u_s)$$

where u_s is the action chosen for state s (either a conditional action in case $s \in CSS$ or a default action in case $s \in CSS'$), and the summations are over the old CSS and CSS' (before updating). [1]

In the measure Ut, we take into account the probabilities of reaching the goal in the future from the current states (based on the Q values; see Theorem 1), as well as the probability of reaching the current states based on the history of the paths traversed (based on $p(s)$'s). This is because what we are aiming at is the estimate of the overall success probability of a path. The basic idea, combining measures of past history and future possibilities , is essentially the same as the A* algorithm. However, instead of an additive combination, we use a multiplicative combination, because probabilities require such combinations. In the algorithm, we select the best actions (for $s \in CSS$ and CSS') that are most likely to succeed based on these measures, respectively.

Note that (1) as a result of incorporating nonconditional default actions, nonconditional plans are special cases of conditional plans. (2) If we set $n = 0$ and

[1] For both CSS and CSS', if a goal state or a state of probability 0 is selected, we may remove it and, optionally, reduce the beam width of the corresponding set by 1.

Action	Description
A0	Pickup
A1	Dry
A2	Paint

Figure 1: The possible actions in the Gripper task.

State Number	Feature Description
0	(not BP, not GC, not HB, not GD)
1	(not BP, not GC, not HB, GD)
2	(not BP, not GC, HB, not GD)
3	(not BP, not GC, HB, GD)
4	(not BP, GC, not HB, not GD)
5	(not BP, GC, not HB, GD)
6	(not BP, GC, HB, not GD)
7	(not BP, GC, HB, GD)
8	(BP, not GC, not HB, not GD)
9	(BP, not GC, not HB, GD)
10	(BP, not GC, HB, not GD)
11	(BP, not GC, HB, GD)
12	(BP, GC, not HB, not GD)
13	(BP, GC, not HB, GD)
14	(BP, GC, HB, not GD)
15	(BP, GC, HB, GD)

Figure 2: The possible states in the Gripper task. State 4 is the starting state. States 14 and 15 are the goal states. The anti-goal states include states 8, 9, 10, and 11. States 0, 1, 2, and 3 are impossible to reach.

$m > 0$, we then in effect have a nonconditional plan extraction algorithm and the result from the algorithm is a nonconditional plan. (3) If we set $m = 0$, then we have a purely conditional plan (with no default action attached).

An issue is how to determine the branching factor, i.e., the number of conditional actions at each step. We can start with a small number, say 1 or 2, and gradually expand the search by adding to the number of branches, until a certain criterion, a termination condition, is satisfied. We can terminate the search when $p(G) > \delta$ (where δ is specified a priori) or when a time limit is reached (in which case failure is declared).

Experiments

An example is the Gripper domain (taken from Kushmerick et al 1995, almost standard in the probabilistic planning literature). In this task, a robot arm has to paint a block without getting itself dirty and to pick up the block in the end. There are three possible actions: paint, dry, and pickup (see Figure 1). There are four binary features describing a state: BP (block painted), GD (gripper dry), HB (holding block), and GC (gripper clean); thus there are a total of 16 states (see Figure 2). Actions have probabilistic outcomes: For example, if the gripper is not dry, it may fail to pick up the block; if the block is painted while the gripper is holding the block, the gripper is likely to get dirty; drying the gripper may not always lead to a dry gripper; and so on. The state transition probabilities, as given in Kushmerick et al (1995), are shown in Figure 3. State 4 is the starting state. States 14 and 15 are the goal states. This task is complex because it involves nondeterministic state transitions, and there are also anti-goals (the states in which the block gets dirty are the anti-goal

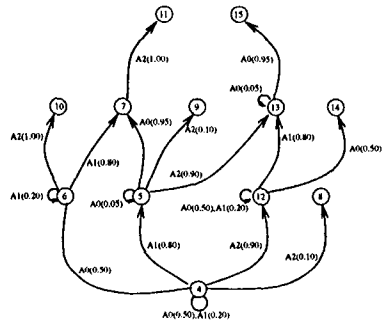

Figure 3: State transitions for the Gripper task, where the states are represented in circles and the state transition probabilities are indicated in parentheses along with the actions that lead to the transitions.

states because there is no possibility of recovery from that; see Figure 2).

We applied Q-learning to learn reactive policies in this task, from interacting with a domain simulator (which was as specified above). Because of the sparcity of possible paths, either discounted or nondiscounted Q-learning worked in this domain. Both were tried and no difference was observed. A conditional plan extracted can be paraphrased as follows:

> Step 1: from the starting state 4, do *paint* (action 2). Step 2: if the current state is 12, do *dry* (action 1); otherwise, the plan fails. Step 3: if the current state is 13, do *pickup* (action 0); otherwise, do *dry* (action 1) again. Step 4: if the current state is not 15, do *pickup* again (action 0). Step 5: if the current state is not 14 or 15, do *pickup* (action 0) again.

A nonconditional plan is simply as follows:

> Step 1: paint. Step 2: dry. Step 3: pickup. Step 4: pickup. [2] Step 5: dry. Step 6: pickup.

Another domain is the Builder task, which is taken from Dearden and Boutilier (1997). In this task, a robot tries to manipulate two objects and join them. There are ten actions that can be performed, including to paint, wash, drill, and shape either object, and to bolt or glue the two together (see Figure 4 for a list of all the actions). Each action has probabilistic outcomes; for example, washing an object may or may not lead to the object being clean. There are a total of nine features that describe a state (including whether the two objects are shaped, drilled, cleaned, painted, and whether they are joined; see Figure 5), which lead to 512 states (but only 330 of these can actually occur). The probabilistic outcomes of actions are given in Dearden and Boutilier (1997). States are numbered

[2] This is done only if previous pickup failed, because otherwise the plan would have been terminated already.

Action	Description
A0	PaintA
A1	PaintB
A2	ShapeA
A3	ShapeB
A4	DrillA
A5	DrillB
A6	WashA
A7	WashB
A8	Bolt
A9	Glue

Figure 4: The possible actions in the Builder task.

Feature Number	Description
F0	A Painted
F1	B Painted
F2	A Shaped
F3	B Shaped
F4	A Drilled
F5	B Drilled
F6	A Clean
F7	B Clean
F8	A and B Joined

Figure 5: The nine features that determine a state in the Builder task.

based on the binary values of the features. Thus, the starting state is state 0, the one in which none of the nine features are true. The goal state is state 511, the one in which all features are true (i.e., the two objects are cleaned, shaped, drilled, painted and joined). The state space of this domain is too large to be pictured here.

We applied Q-learning to this task, through interacting with the domain simulator. The task was nondeterministic (each action had probabilistic outcomes) and had a larger state space. However, whereas the gripper task had anti-goal states, this task did not, and the ordering of the actions to be performed was not as important. Because of the sparcity of possible paths, both discounted or nondiscounted Q-learning worked in this domain. One resulting conditional plan is shown in Figure 6.

Yet another domain is the simulated minefield navigation task developed by NRL (see Gordon et al 1994). In the task as shown in Figure 7, the agent has to navigate an underwater vessel through a minefield to reach a target location. The agent receives information only from instruments. As shown in Figure 8, the sonar gauge shows how close the mines are in 7 equal areas that range from 45 degrees to the left of the agent to 45 degrees to the right. The fuel gauge shows how much time is left before fuel runs out. The bearing gauge shows the direction of the target from the present direction of the agent. The range gauge shows how far the target is from the current location. Based only on such information, the agent decides on (1) how to turn and (2) how fast to move. The agent, within an allotted time period, can either (a) reach the target (a success), (b) hit a mine (a failure), or (c) run out of fuel (a failure again). A random mine layout is generated for each episode. The time allotted to the agent for each episode is 200 steps. There are 41 inputs and thus more than 10^{12} (input) states. Mathematically, it

```
Step 0: [( , A2)]
Step 1: [(F2, A4) ( , A2)]
Step 2: [(F2F4, A5) (F2, A4) ( , A2)]
Step 3: [(F2F4F5, A3) (F2F4, A5) (F2, A4)]
Step 4: [(F2F3F4F5, A6) (F2F4F5, A3) (F2F4, A5)]
Step 5: [(F2F3F4F5F6, A8) (F2F3F4F5, A6) (F2F4F5, A3) (F2F4, A5)]
Step 6: [(F2F3F4F5F6, A8) (F2F3F4F5F6F8, A7) (F2F3F4F5, A6) (F2F4F5, A3) (F2F4, A5)]
Step 7: [(F2F3F4F5F6F8, A7) (F2F3F4F5F6F7F8, A1) (F2F3F4F5F6, A8) (F2F3F4F5, A6) (F2F4F
Step 8: [(F2F3F4F5F6F7F8, A1) (F2F3F4F5F6F8, A7) (F1F2F3F4F5F6F7F8, A0) (F2F3F4F5F6, A8
Step 9: [(F2F3F4F5F6F7F8, A1) (F1F2F3F4F5F6F7F8, A0) (F2F3F4F5F6F8, A7) (F1F2F3F4F5F7F8
Step 10: [(F1F2F3F4F5F6F7F8, A0) (F2F3F4F5F6F7F8, A1) (F2F3F4F5F6F8, A7) (F2F3F4F5F7F8,
Step 11: [(F2F3F4F5F6F7F8, A1) (F1F2F3F4F5F6F7F8, A0) (F2F3F4F5F7F8, A6) (F1F2F3F4F5F7F
Step 12: [(F1F2F3F4F5F6F7F8, A0) (F2F3F4F5F6F7F8, A1) (F2F3F4F5F7F8, A6) (F2F3F4F5F7F8,
Step 13: [(F1F2F3F4F5F6F7F8, A0) (F2F3F4F5F6F7F8, A1) (F2F3F4F5F7F8, A6) (F1F2F3F4F5F7F
Step 14: [(F1F2F3F4F5F6F7F8, A0) (F2F3F4F5F6F7F8, A1) (F2F3F4F5F7F8, A6) (F1F2F3F4F5F6F
Step 15: [(F2F3F4F5F6F7F8, A1) (F1F2F3F4F5F6F7F8, A0) (F2F3F4F5F7F8, A6) (F1F2F3F4F5F7F
Step 16: [(F1F2F3F4F5F6F7F8, A0) (F2F3F4F5F6F7F8, A1) (F2F3F4F5F7F8, A6)]
Step 17: [(F1F2F3F4F5F6F7F8, A0) (F2F3F4F5F6F7F8, A1)]
Step 18: [(F1F2F3F4F5F6F7F8, A0)]
```

Figure 6: A conditional plan extracted for the Builder task. $n = 6, m = 8$. In each pair of parentheses, a state is specified by the set of positive features (while other features are negative), which is followed by an action. Action and feature labels are described in Figures 4 and 5. Default actions were not utilized in this plan, because CSS' was almost always empty.

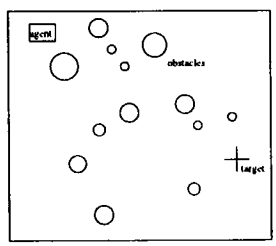

Figure 7: Navigating through a minefield

is a partially observable Markov decision process (i.e., POMDP; Bertsekas and Tsitsiklis 1996, Chrisman 1993, McCallum 1996, Kaelbling et al 1996).

Because in this domain there were many possible paths that were vastly different in length, in an attempt to obtain the optimal (shortest) paths to the goal, we adopted discounted Q-learning and Q-learning with step penalty to encourage shorter paths. Because of the large state space (due to the large number of input dimensions in this task), we used state aggregation and approximation methods in the form of a backprop-

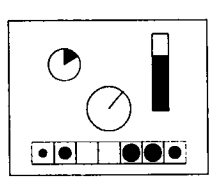

Figure 8: The Navigation Input
The display at the upper left corner is the fuel gauge; the vertical one at the upper right corner is the range gauge; the round one in the middle is the bearing gauge; the 7 sonar gauges are at the bottom.

agation neural network for model estimation (see Sun 1997). For learning Q values, we also used an approximator in the form of a backpropagation neural network.

We applied Q-learning and model estimation in the above forms to the simulator for this domain (Gordon et al 1994). Based on the results of Q-learning, a plan was extracted. Basically, it indicates a strategy for navigating through the minefield: avoids mines and goes in the general direction of the target. The estimated success rate (the lower bound) of the plan was 62% as opposed to the success rate of the original Q-learner of 81%

The advantage of the extracted plans is "policy compression": While the original policy requires the specification of the best actions for each of these states, the extracted plans require far less specifications. The plans (conditional or nonconditional) require the specification of less than 100 actions. Thus the plans save representational cost. The extracted plans also save sensing cost, because it can be executed with less sensing (for conditional plans) or no sensing at all (for nonconditional plans), assuming sensing consumes computational resources in proportion to the number of states to be distinguished.

Discussions

Compared with Kushmerick et al (1995), Dearden and Boutilier (1996), Maclin and Shavlik (1994) and other work of planning with DP/RL, the difference is obvious: our approach is bottom-up (turning Q values from DP/RL into explicit planning rules) while their approaches were top-down (i.e., using given rules and turning them into MDP for DP/RL; see section 1). Top-down approaches require much a priori domain knowledge to begin with, which may not be available. On the other hand, compared with pure DP/RL (such as Watkins 1989, Barto et al 1995, Sutton 1990), we extract explicit plans while their models produced only reactive policies and did not provide open-loop or semi-open-loop policies (i.e., explicit plans).

The key insight underlying our plan extraction algorithms is the relation between Q values and the probabilities of reaching goals in the future. Based on that, temporal projection search is used in our algorithm to extract plans.

Kushmerick et al (1995) and Penbenthy and Weld (1992) also used explicit search (temporal projection) in planning. The complexity of Kushmerick et al (1995) and Penbenthy and Weld (1992) was clearly exponential. Our method is more efficient, because it first learns implicitly through using DP/RL algorithms (which learns implicit sequencing), the cost of which is comparable to, or less than, the cost of collecting statistics needed in Kushmerick et al (1995) (the process of which was not specified in Kushmerick et al 1995 and is thus assumed to be random). Instead of collecting statistics only, we acquire Q values as well as collecting statistics with the use of DP/RL that facilitates the process. Then, some limited search (beam search) is conducted in our method to extract explicit plans. The cost of the search is relatively limited due to the limit on the beam width, which is made possible by using the guidance provided by the Q functions (acquired through learning with DP/RL algorithms). A Q function provides estimates of the future promises, or success probabilities, of different paths. On that basis, our algorithm narrows down the search, focusing only on the most promising paths, and avoids an exhaustive search of the entire space of all possible paths.

Some recent work in robotics used temporal projection to extract plans, such as Tani (1996). However, Tani (1996) used gradient descent on a cost function (which encourages shorter paths to goals). Notably, there was no guarantee of soundness or completeness for such an approach. His work did not deal with probabilistic worlds, which posed a serious difficulty for the method.

Conclusions

In sum, this paper presented a two-stage approach for learning to plan by first acquiring closed-loop policies through DP/RL and by subsequently extracting open-loop or semi-open-loop policies from the learned closed-loop policies. The novelty of the work lies in this two-step process and the algorithms that go from the first step (acquiring closed-loop policies) to the second step (acquiring open-loop or semi-open-loop policies). The work addressed the theoretical foundation of this process, in ways of interpreting Q values acquired from DP/RL as probabilities of achieving goals. A beam search algorithm utilizing A*-like heuristics was devised to perform this extraction process.

Acknowledgements: This work is supported in part by Office of Naval Research grant N00014-95-1-0440. The simulator for the navigation task was provided by NRL (through Diana Gordon, Jim Ballas and Alan Schultz).

References

P. Agre and D. Chapman, (1990). What are plans for? In: P. Maes, (ed.) *Designing Autonomous Agents.* Elsevier, New York.

A. Barto, S. Bradtke, and S. Singh, (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence.* 72 (1-2), 81-138.

R. Bellman, (1957). *Dynamic Programming.* Princeton University Press, Princeton, NJ. D. Bertsekas and D. Castanon, (1989). Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transaction on Automatic Control.* 34, 6. 589-598.

D. Bertsekas and J. Tsitsiklis, (1996). *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA.

R. Dearden and Boutilier, (1996). Abstraction and approximate decision theoretic planning. *Artificial Intel-*

ligence. 89, 219-283.

D. Gordon, et al. (1994). NRL task: navigation and collision avoidance. Naval Research Lab. Washington, DC.

L. Kaelbling, M. Littman, and A. Moore, (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 237-285.

N. Kushmerick, S. Hanks and D. Weld, (1995). An algorithm for probabilistic planning. *Artificial Intelligence.*

R. Maclin and J. Shavlik, (1994). Incorporating advice into agents that learn from reinforcements. *Proc.of AAAI-94.* Morgan Kaufmann, San Meteo, CA.

J. Penbenthy and D. Weld, (1992). UCPOP: a sound, complete, partial order planner for ADL. *Proc. 3rd International Conference on Knowledge Representationa and Reasoning.* 103-114.

M. Pollack, (1992). The use of plans. *Artificial Intelligence*, 57, 43-68.

E. Sacerdoti, (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence.* 5, 115-135.

M. Schoppers, (1987). Universal plans for reactive robots in unpredictable environments. *Proc. of IJCAI*, pp.1039-1046.

R. Sun, (1997). Planning from reinforcement learning. Technical report TR-CS-97-0027, University of Alabama, Tuscaloosa, AL.

R. Sun and T. Peterson, (1997). A hybrid model for learning sequential navigation. *Proc. of IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA'97).* Monterey, CA. pp.234-239. IEEE Press.

R. Sutton, (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proc.of Seventh International Conference on Machine Learning.*

J. Tani, (1996). Model-based learning for mobile robot navigation from the dynamic systems perspective, *IEEE Transaction on Systems, Man and Cybernetics – Part B: Cybernetics,* 26, 3, 421-436.

D. Warren, (1976). Generating conditional plans and programs. In *Proc. of AISB Summer Conference.* 344-354. University of Edinburgh, UK.

C. Watkins, (1989). *Learning with Delayed Rewards.* Ph.D Thesis, Cambridge University, Cambridge, UK.