Learning policies for partially observable environments: Scaling up

Michael L. Littman
mlittman@cs.brown.edu

Anthony R. Cassandra
arc@cs.brown.edu

Leslie Pack Kaelbling
lpk@cs.brown.edu

Department of Computer Science
Brown University
Providence, RI 02912-1910

Abstract

Partially observable Markov decision processes (POMDP's) model decision problems in which an agent tries to maximize its reward in the face of limited and/or noisy sensor feedback. While the study of POMDP's is motivated by a need to address realistic problems, existing techniques for finding optimal behavior do not appear to scale well and have been unable to find satisfactory policies for problems with more than a dozen states. After a brief review of POMDP's, this paper discusses several simple solution methods and shows that all are capable of finding near-optimal policies for a selection of extremely small POMDP's taken from the learning literature. In contrast, we show that none are able to solve a slightly larger and noisier problem based on robot navigation. We find that a combination of two novel approaches performs well on these problems and suggest methods for scaling to even larger and more complicated domains.

1 INTRODUCTION

Mobile robots must act on the basis of their current and previous sensor readings. In spite of improvements in technology, a robot's information about its surroundings is necessarily incomplete: sensors are imperfect, objects occlude one another from view, the robot might not know its initial status or precisely where it is. The theory of *partially observable Markov decision processes* (POMDP's) (Astrom, 1965; Smallwood and Sondik, 1973; Cassandra et al., 1994) models this situation and provides a basis for computing optimal behavior.

A variety of algorithms have been developed for solving POMDP's (Lovejoy, 1991), but because the problem is so computationally challenging (Papadimitriou and Tsitsiklis, 1987), most techniques are too inefficient to be used on all but the smallest problems (2 to 5 states (Cheng, 1988)). Recently, the Witness algorithm (Cassandra, 1994; Littman, 1994) has been used to solve POMDP's with up to 16 states. While this problem size is considerably larger than prior state of the art, the algorithm is not efficient enough to be used for larger POMDP's.

Thus, the generality and expressiveness of the POMDP framework comes with a cost: only extremely small problems can be solved using available techniques. This paper is an incremental attempt at narrowing the gap between promise and practice. Using reinforcement-learning techniques and insights from the POMDP literature, we show how a satisfactory policy can be found for a POMDP with close to 100 states and dozens of observations.

We assume that a complete and accurate model of the state transition dynamics is given and use various techniques to construct a policy that achieves high reward. Even with these restrictions, the problem of finding optimal behavior is still too difficult and we have chosen to simplify it in several respects. First, we will be satisfied if we can find reasonably good suboptimal policies. Secondly, our training and testing is done using simulated runs from a fixed initial distribution, limiting the set of situations for which the algorithms need to find good behavior.

The structure of the paper is as follows. The introduction summarizes formal results concerning the POMDP model. The next section describes several methods for finding approximately optimal policies and provides evidence that all perform comparably on a collection of extremely small problems. Of these, a simple approach based on solving the underlying MDP is clearly the most time efficient. None of these approaches can solve two slightly larger navigation problems and so the next section presents a more successful hybrid approach that seeds learning using the Q values of the underlying MDP. The concluding section considers a class of problems that require a richer representation for policies and presents preliminary results on a technique for learning such policies.

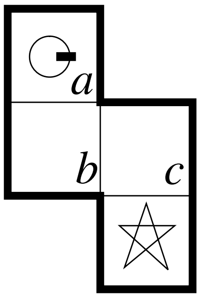

Figure 1: A tiny navigation environment.

2 PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

This section reviews the operations research literature on POMDP's.

2.1 DEFINITIONS AND EXAMPLE

A POMDP is a tuple $< S, A, T, R, O, \Omega >$ where S is a set of states, A a set of actions, and Ω a set of observations. We will only consider the case in which these sets are finite.

The functions T and R define a Markov decision process (MDP) (Bertsekas, 1987) with which the agent interacts without direct information as to the current state. The transition function, $T : S \times A \rightarrow \Pi(S)$, specifies how the various actions affect the state of the environment. ($\Pi(\cdot)$ represents the set of discrete probability distributions over a finite set.) The agent's immediate rewards are given by $R : S \times A \rightarrow \Re$. The agent's decisions are made based on information from its sensors (observations) formalized by $O : S \times A \rightarrow \Pi(\Omega)$.

Our goal in this work is to take a POMDP and find a *policy*, which is a strategy for selecting actions based on the information available to the agent, that maximizes an infinite-horizon, discounted optimality criterion.

Figure 1 depicts a tiny navigation POMDP that we use for explanatory purposes. It consists of 13 states (4 possible orientations in each of 3 rooms and a goal state which is denoted by a star), 9 observations (relative location of the surrounding walls, plus "star"), and 3 actions (forward, rotate left, rotate right). The problem is intended to model a robot in a simple office environment. In the figure, the robot symbol occupies the "East in Room a" state. The agent's task is to enter the room marked with the star, at which point it receives a reward of +1. After receiving the reward, the agent's next action transports it at random into one of the 12 non-goal states. Otherwise, transitions and observations are deterministic in this example.

2.2 THE BELIEF MDP

In the tiny navigation environment, the immediate observations do not supply enough information for the agent to disambiguate its location nor are they sufficient for indicating the agent's best choice of action. For example, if the agent sees a wall behind it and to its left, it might be in "North in Room b" (optimal action is to turn right) or "South in Room c" (optimal action is to go forward to the goal).

Some form of memory is necessary in order for our agent to choose its actions well. Although many architectures are possible, one elegant choice is to maintain a probability distribution over the states of the underlying environment. We call these distributions *belief states* and use the notation $b(s)$ to indicate the agent's belief that it is in state s when the current belief state is $b \in \Pi(S)$. Using the model, belief states can be updated based on the agent's actions and observations in a way that makes the beliefs correspond exactly to state occupation probabilities.

From a known starting belief state, it is easy to use the transition and observation probabilities to incorporate new information into the belief state (Cassandra et al., 1994). As an example, consider an agent that is started in any of the 12 non-goal states of the tiny navigation environment with equal probability: $b(s) = 1/12$ for all non-goal states. If the agent chooses to turn right and then sees walls in front of it and to its right, only two states are possible:

$$b(\text{ South in Room } b\ ) = b(\text{ North in Room } c\ ) = 1/2\ .$$

After next moving forward and seeing walls in all directions except behind, the agent is sure of where it is:

$$b(\text{ North in Room } a\ ) = 1.$$

Since the agent's belief state is an accurate summary of all the relevant past information, it is a *sufficient statistic* for choosing optimal actions (Bertsekas, 1987). That is, an agent that can choose the optimal action for any given belief state is acting optimally in the environment.

An important consequence is that the belief states, in combination with the updating rule, form a completely observable Markov decision process (MDP) with a continuous state space, similar to problems addressed in the reinforcement-learning literature (Moore, 1994). Our goal will be to find an approximation of the Q function over the continuous space of belief states and to use this as a basis for action in the environment. We restrict our attention to stationary, deterministic policies on the belief state, since this class is relatively simple and we are assured that it includes an optimal policy (Ross, 1983).

2.3 PIECEWISE-LINEAR CONVEX FUNCTIONS

A particularly powerful result of Sondik's is that the optimal value function for any POMDP can be approximated arbitrarily well by a piecewise-linear and convex (PWLC) function (Smallwood and Sondik, 1973; Littman, 1994). Further, there is a class of POMDP's that have value functions that are exactly PWLC (Sondik, 1978). These results apply to the optimal Q functions as well: the Q function for action a, $Q_a(b)$ is the expected reward for a policy that starts in belief state b, takes action a, and then behaves optimally. By choosing the action that has the largest Q value for a given belief state, an agent can behave optimally.

PWLC functions are particularly convenient because of their representational simplicity. If $Q_a(b)$ is a PWLC function, then $Q_a(b)$ can be written:

$$Q_a(b) := \max_{q \in L_a} q \cdot b$$

for some finite set of $|S|$-dimensional vectors, L_a. That is, Q_a is just the maximum of a finite set of linear functions of b.

So, although we are trying to find a solution to a continuous-space MDP, we have constraints on the form of the optimal Q functions that make this search a great deal simpler.

3 SOME SOLUTION METHODS FOR POMDP's

This section sketches several methods for finding linear or PWLC approximations to the optimal Q functions for POMDP's. The goal in each of them is to find Q functions that can be used to generate good behavior; that is, we will judge the methods by the policies they produce and not by the accuracy with which they estimate the optimal Q values. None of these methods are entirely original, but none have been used to find fast approximations to optimal policies for POMDP's given the POMDP models.

3.1 TRUNCATED EXACT VALUE ITERATION

The Witness algorithm (Cassandra et al., 1994; Littman, 1994) finds exact solutions to discounted finite-horizon POMDP's using value iteration. After its k-th iteration, the algorithm returns the exact k-step Q functions as collections of vectors, L_a, for each action, a. The algorithm can be used to find arbitrarily accurate approximations to the optimal infinite-horizon Q functions and therefore policies that are arbitrarily close to optimal (Williams and Baird, 1993). Unfortunately, the algorithm can take many, many iterations to find an approximately optimal value function, and for problems with a large number of observations, the size of the L_a sets can grow explosively from iteration to iteration. Nonetheless, it is often the case that a near-optimal policy is reached long before the Q values have converged to their optimal values, so truncating the value iteration process prematurely can still yield excellent policies. We call this approach "truncated exact value iteration" and denote it as Trunc-VI.

3.2 THE Q_{MDP} VALUE METHOD

Another natural approach to finding Q functions for POMDP's is to make use of the Q values of the underlying MDP. That is, we can temporarily ignore the observation model and find the $Q_{\mathrm{MDP}}(s, a)$ values for the MDP consisting of the transitions and rewards only. These values can be computed extremely efficiently for problems with dozens to thousands of states and a variety of approaches are available (Puterman, 1994).

With the Q_{MDP} values in hand, we can treat all the Q_{MDP} values for each action as a single linear function and estimate the Q value for a belief state b as $Q_a(b) = \sum_s b(s)\; Q_{\mathrm{MDP}}(s, a)$. This estimate amounts to assuming that any uncertainty in the agent's current belief state will be gone after the next action. Thus, the action whose long-term reward from all states (weighted by the probability of occupying the state) is largest will be the one chosen at each step.

Policies based on this approach can be remarkably effective. One drawback, though, is that these policies will not take actions to gain information. For instance, a "look around without moving" action and a "stay in place and ignore everything" action would be indistinguishable with regard to the performance of policies under an assumption of one-step uncertainty. This can lead to situations in which the agent loops forever without changing belief state.

3.3 REPLICATED Q-LEARNING

Chrisman (1992) and McCallum (1992) explored the problem of learning a POMDP model in a reinforcement-learning setting. At the same time that their algorithms attempt to learn the transition and observation probabilities, they used an extension of Q-learning (Watkins, 1989) to learn approximate Q functions for the learned POMDP model. Although it was not the emphasis of their work, their "replicated Q-learning" rule is of independent interest.

Replicated Q-learning generalizes Q-learning to apply to vector-valued states and uses a single vector, q_a, to approximate the Q function for each action a: $Q_a(b) = q_a \cdot b$. For many POMDP's, a single vector per action is not sufficient for representing the optimal policy. Nonetheless, this approximation is simple and can be remarkably effective.

The components of the vectors are updated using

$$\Delta q_a(s) = \alpha\ b(s)(r + \gamma \max_{a'} Q_{a'}(b') - q_a(s)) \ .$$

The update rule is evaluated for every $s \in S$ each time the agent makes a state transition; α is a learning rate, b a belief state, a the action taken, r the reward received, and b' the resulting belief state. This rule applies the Q-learning update rule to each component of q_a in proportion to the probability that the agent is currently occupying the state associated with that component.

By simulating a series of transitions from belief state to belief state and applying the update rule at each step, this learning rule can be used to solve a POMDP. If the observations of the POMDP are sufficient to ensure that the agent is always certain of its state (i.e., $b(s) = 1$ for some s at all times), this rule reduces exactly to standard Q-learning and can be shown to converge to the optimal Q function under the proper conditions (Jaakkola et al., 1994; Tsitsikilis, 1994).

The rule itself is an extremely natural extension of Q-learning to vector-valued state spaces, since it basically consists of applying the Q-learning rule at every state where the magnitude of the change of a state's value is proportional to the probability the agent is in that state. In fact, in addition to its use by Chrisman and McCallum, an elaboration of this rule is used by Connell and Mahadevan (1993) for solving a distributed-representation reinforcement-learning problem.

Although replicated Q-learning is a generalization of Q-learning, it does not extend correctly to cases in which the agent is faced with significant uncertainty. Consider a POMDP in which the optimal Q function can be represented with a single linear function. Since replicated Q-learning independently adjusts each component to predict the moment-to-moment Q values, the learning rule will tend to move all the components of q_a toward the same value.

3.4 LINEAR Q-LEARNING

Linear Q-learning is extremely similar to replicated Q-learning but instead of training each component of q_a toward the same value, the components of q_a are adjusted to match the coefficients of the linear function that predicts the Q values. This is accomplished by applying the delta rule for neural networks (Rumelhart et al., 1986), which, adapted to the belief MDP framework, becomes:

$$\Delta q_a(s) = \alpha\ b(s)(r + \gamma \max_{a'} Q_{a'}(b') - q_a \cdot b) \ .$$

Like the replicated Q-learning rule, this rule reduces to ordinary Q-learning when the belief state is deterministic.

In neural network terminology, linear Q-learning views $\{b,\ r + \gamma \max_{a'} Q_{a'}(b')\}$ as a training instance for the function $Q_a(\cdot)$. Replicated Q-learning, in contrast, uses this example as a training instance for the component $q_a(s)$ for every s. We should expect the rules to behave differently when the components of q_a need to have widely different values to solve the problem at hand.

Like replicated Q-learning, linear Q-learning has the limitation that only linear approximations to the optimal Q functions are considered. In general, this can lead to policies that are arbitrarily poor, although this does not appear to be true for the extremely small POMDP's we studied.

Note that, since the transition probabilities and rewards are known, it is possible to perform full backups instead of the sampled backups used in traditional Q-learning. Our preliminary experiments indicate that full backups do not appear to speed convergence (at least not consistently across POMDP's) and require significant computational overhead (Littman et al., 1995). More study will be necessary to fully address this issue. All of the results reported here use sample backups.

3.5 EMPIRICAL COMPARISON ON EXTREMELY SMALL PROBLEMS

We ran each of the above methods on a battery of POMDP's selected from the literature, summarized in Table 1. The details of the problems are not crucial and there is not space here to describe them—the reader is referred to the appropriate references for descriptions.

Interestingly, all 6 POMDP's have the property that optimal policies periodically reset to a problem-specific belief state. We used a discount factor of 0.95 for all problems. The column of Table 1 labeled "Noise" indicates whether there is noise in the transitions, observations, or both. The part-painting problem has been adapted from its original form (Littman et al., 1995). The 4x3 grid problem was introduced by Russell and Norvig (1994) and the version here includes a discounted criterion and returns to the initial belief state after a goal instead of entering an absorbing state.

For the experiments on truncated exact value iteration, we ran the exact algorithm for approximately 100 seconds and used the output of the last complete iteration as a solution.

The learning approaches have a large number of free parameters which we did not optimize carefully for either speed or performance. For each of 21 runs, we performed 75,000 steps of learning starting from the problem-specific belief state. During learning, actions were selected to maximize the current Q functions with

Name	$\|S\|$	$\|A\|$	$\|\Omega\|$	Noise
Shuttle (Chrisman, 1992)	8	3	5	T/O
Cheese Maze (McCallum, 1992)	11	4	7	–
Part Painting (Kushmerick et al., 1993)	4	4	2	T/O
4x4 Grid (Cassandra et al., 1994)	16	4	2	–
Tiger (Cassandra et al., 1994)	2	3	2	O
4x3 Grid (Parr and Russell, 1995)	11	4	6	T

Table 1: A suite of extremely small POMDP's.

	Shuttle	Cheese Maze	Part Painting	4x4 Grid	Tiger	4x3 Grid
Trunc VI	1.805 ± 0.014	0.188 ± 0.002	0.179 ± 0.012	0.193 ± 0.003	0.930 ± 0.205	0.109 ± 0.005
Q_{MDP}	1.809 ± 0.012	0.185 ± 0.002	0.112 ± 0.016	0.192 ± 0.003	1.106 ± 0.196	0.112 ± 0.005
Repl Q	1.355 ± 0.265	0.175 ± 0.017	0.003 ± 0.005	0.179 ± 0.013	1.068 ± 0.047	0.080 ± 0.014
Linear Q	1.672 ± 0.121	0.186 ± 0.000	0.132 ± 0.030	0.141 ± 0.026	1.074 ± 0.046	0.095 ± 0.007
optimal	—	0.186 ± 0.002	0.170 ± 0.012	0.192 ± 0.002	1.041 ± 0.180	—

Table 2: Results of POMDP solution methods on the suite of extremely small problems.

a 0.10 probability of being overridden by a uniform random action. The learning rate was decreased according to the following schedule: 0.1 for steps 0 to 20,000, 0.01 from 20,000 to 40,000, 0.001 from 40,000 to 60,000, and then 0.0001 thereafter. The $q_a(s)$ component values were initialized to random numbers uniformly chosen between -20.0 and 20.0. The parameter values were chosen by informally monitoring the performance of linear Q-learning on several of the problems.

Each method returned a set of vectors that constitute linear or PWLC approximations of the Q functions. An agent that chooses actions to maximize the Q functions was then simulated to evaluate the quality of the induced policy. Each simulation started with the agent in the problem-specific belief state and ran for 101 steps. This procedure was repeated 101 times and the performance is reported as the mean reward received with a 95% confidence interval.

Table 2 reports the results. The data for the two learning algorithms are pooled over 21 independent experiments. For four of the problems, we were able to compute the optimal Q functions using the Witness algorithm in 25 to 120 minutes. We then simulated the optimal vectors to obtain the row marked "optimal" in the table. The two other problems possibly do not have PWLC optimal Q functions.

The most overwhelming result is that almost every method on almost every problem achieves practically optimal performance. Truncated exact value iteration is always statistically indistinguishable from optimal and tends to do no worse than the Q_{MDP} value method. The Q_{MDP} value method tends to do no worse than linear Q-learning which tends to do no worse than replicated Q-learning. The Q_{MDP} value method, which consistently performed quite well, was the most time-efficient algorithm, requiring no more than half a second on any problem. The learning algorithms, by contrast, took between 16 seconds and 80 seconds, depending mostly on the size of the problem. The truncated exact value iteration algorithm always took 100 seconds, by design.

There are two significant exceptions to the overall trend mentioned above: the Q_{MDP} value method was worse than linear Q-learning on the part-painting problem and linear Q-learning was worse than replicated Q-learning on the 4x4 problem. The former is a result of the Q_{MDP} value method not choosing actions to gain information, which are necessary for optimal behavior in this problem. The latter occurs because of the determinism in the state transitions and the relatively small probability of taking random actions; this problem can be easily fixed by adjusting the random-action probability (Littman et al., 1995). This combination can cause the goal to be infrequently visited during learning in cases where the random initial policy leads to cyclic behavior.

4 HANDLING LARGER POMDP's: A HYBRID APPROACH

It is worth asking whether the results of the previous section apply to larger or more complicated domains. We constructed two POMDP's designed to model a robot navigation domain, shown in Figures 2 and 3.

One environment has 57 states (14 rooms with 4 orientations each, plus a goal) and 21 observations (each possible combination of the presence of a wall in each of the 4 relative directions, plus "star" and three landmarks visible when the agent faces south in three particular locations). The other has 89 states (4 orientations in 22 rooms, plus a goal) and 17 observations (all combinations of walls, plus "star"). Both include 5 actions (stay in place, move forward, turn right, turn

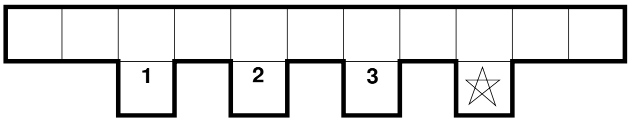

Figure 2: Navigation environment with 57 states.

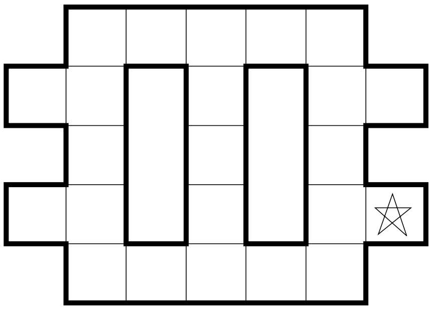

Figure 3: Navigation environment with 89 states.

left, turn around) and have extremely noisy transitions and observations (Littman et al., 1995).

We ran the same collection of algorithms on these two environments with a slight change: truncated exact value iteration was given roughly 1000 seconds. Performance was measured slightly differently. The policies were evaluated for 251 trials, each consisting of a run from a problem-specific initial belief state to the goal. For these two environments the initial belief state was a uniform distribution over all states except the goal state. If the agent was unable to reach the goal in 251 steps, the trial was terminated.

Table 3 reports the percentage of the 251 runs in which the agent reached the goal and the median number of steps to goal over all 251 runs. For the learning algorithms, performance was measured as a median of 21 independent runs.

This time, none of the approaches gave even passable results, with many test runs never reaching the goal after hundreds of steps. Truncated exact value iteration was able to complete two iterations in about 4 seconds and made no additional progress for up to 1500 seconds. The Q_{MDP} value method is deterministic, so the reported results are based on the best policy it can achieve. The learning approaches have the capability

	57 states		89 states	
	goal%	median	goal%	median
Trunc VI	62.9	150	44.6	> 251
Q_{MDP}	47.4	> 251	25.9	> 251
Repl Q	5.2	> 251	2.8	> 251
Linear Q	8.4	> 251	5.2	> 251

Table 3: Results of POMDP solution methods on the two navigation environments.

	57 states		89 states	
	goal%	median	goal%	median
Repl Q	72.9	21	10.8	> 251
Linear Q	96.0	15	58.6	51
Human	100.0	15	100.0	29
Q_{MDP}-no stay	100.0	16	57.8	40
Random Walk	46.2	> 251	25.9	> 251

Table 4: Results of POMDP solution methods when seeded with the Q_{MDP} values on two navigation environments.

of adapting and improving but are unable to reach the goal state often enough to learn anything at all. Thus, all 4 methods fail, but for different reasons.

This suggests the possibility of a hybrid solution. By computing the Q_{MDP} values and using them to seed the q_a vectors for learning, we can take advantage of the strengths of both approaches. In particular, the hope is that the Q_{MDP} values can be computed quickly and then improved by the learning algorithms.

Table 4 summarizes the results of initializing the two learning algorithms using the Q_{MDP} values in place of random vectors. Training and testing procedures followed those of the other navigation experiments.

In both environments, the linear Q-learning algorithm was able to use the initial seed values to find a better policy (almost doubling the completion percentage and halving the steps to the goal). The replicated Q-learning algorithm, on the other hand, actually made the performance of the Q_{MDP} value method worse.

The performance of the hybrid algorithm appears quite good. However, the complexity of the navigation environments makes direct comparison with an optimal policy out of the question. To get a qualitative sense of the difficulty, we created an interactive simulator for the two navigation environments which included a graphical belief state display. A single human subject (one of the authors) practiced using the simulator and then carried out testing trials with the results reported in Table 4. In the smaller environment, the testing period lasted for 45 trials and the longest run was 57 steps. The median performance of 15 steps per trial is exactly the same as that of the hybrid algorithm. In the larger environment, the testing period lasted for 31 trials and the longest run was 73 steps indicating substantial room for improvement in the existing algorithms.

After further study, we discovered that the primary reason for the poor performance of the straight Q_{MDP} value method is that the agent chooses the "stay in place" action in some belief states and sometimes becomes trapped in a cycle. As a test of this hypothesis, we removed this action from the set of actions that can be chosen by the Q_{MDP} value method and reran the evaluation with results given in Table 4. Surpris-

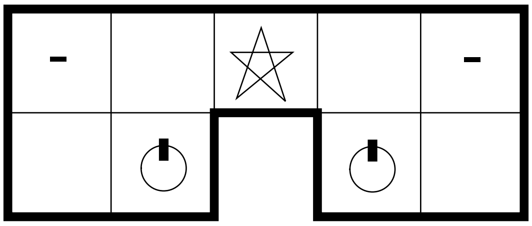

Figure 4: A 33-state navigation environment that cannot be solved with a single linear function per action.

ingly, decreasing the set of options *helped* the Q_{MDP} value method reach a level of performance comparable to that of linear Q-learning. Thus, the learning algorithm applied to the navigation environments may be retaining the important parts of the Q_{MDP} policy while simply learning to suppress the "stay in place" action—a reasonable approach to attaining good performance on these POMDP's. For comparison purposes, we have included the performance of a random walk policy where actions (except "stay in place") are chosen randomly.

Seeding linear Q-learning using the Q_{MDP} values leads to a promising method of solving larger POMDP's than have been addressed to date. More study is needed to understand the strengths and limitations of this approach.

5 MORE ADVANCED REPRESENTATIONS

None of the algorithms reach the goal in the 89-state problem all the time: clearly optimal performance has not yet been reached. As discussed in Section 2.3, piecewise-linear convex functions can approximate the optimal Q functions as closely as necessary. In contrast, the linear functions used by the learning algorithms can result in arbitrarily bad approximations.

5.1 THE NEED FOR A MORE ADVANCED REPRESENTATION

To drive this point home, we designed a navigation problem (see Figure 4) for which any linear approximation to the Q functions is guaranteed to be suboptimal. The parameters of the environment follow those of the navigation environments discussed previously. There are two significant differences: the two rooms marked with minus signs in the figure are associated with negative reward, and the agent starts with equal probability facing North in one or the other of the two rooms marked with robot symbols in the figure.

An agent starting in the left start state should move forward, turn right, and move forward again. From the right start state, the agent should move forward, turn *left* and move forward again. The difficulty is that the two scenarios are distinguished *only* by the configuration of walls in the initial state, which can only be perceived if the agents chooses to stay in place for a step so that it may receive an observation for the initial state. Because actions precede observations, staying in place is an action to gain information in this problem.

The fact that the agent needs to take an action to gain information and then execute the same action (forward) regardless of the outcome, is sufficient to destroy any single-vector-per-action approximation of the optimal policy (Littman et al., 1995). Although we understand the nature of this particular problem, a very interesting (and open) problem is how to determine the number of vectors needed to represent the optimal policy for any given POMDP.

5.2 A PWLC Q-LEARNING ALGORITHM

A simple approach to learning a PWLC Q function is to maintain a set of vectors for each action and use a competitive updating rule: when a new training instance (i.e., belief state/value pair) arrives, the vector with the largest dot product is selected for updating. The actual update follows the linear Q-learning rule. It is possible that the different vectors will come to cover different parts of the state space and thereby represent a more complex function than is possible with a single vector.

To show the potential gain of utilizing multiple vectors per action, we ran experiments on the 33-state navigation environment. We ran 21 independent trials of 75,000 learning steps of linear Q-learning as well as truncated exact value iteration and the Q_{MDP} value method. We compared these to the 3-PWLC Q-learning algorithm, which uses the competitive approach described above with 3 vectors per action. In analogy to the hybrid algorithm of the previous section, we initialize all 3 vectors for each action with the appropriate Q_{MDP} values.

The evaluation criterion was the same as for the 57 and 89-state navigation environment experiments. Table 5 shows the results and, as anticipated, the single vector methods perform poorly.

Although the 3-PWLC algorithm performs astonishingly well on this problem, its performance on other problems has been inconsistent. The primary difficulty is that noisy updates can cause a vector to "sink" below the other vectors. Since this approach only updates vectors when they are the largest for some belief state, these sunken vectors can never be recovered. A related problem plagues almost all competitive learning methods and in our informal experiments, we found this to occur quite often. We have considered some extensions to address this problem, but we have not yet found a reliable solution.

	33 states	
	goal%	median
trunc VI	39.8	> 251
Q_{MDP}	17.9	> 251
Linear Q	46.6	> 251
3-PWLC Q	98.4	5
Q_{MDP}-no stay	14.3	> 251

Table 5: Results of POMDP solution methods on the specially-constructed 33-state navigation environment.

A classic approach to the sunken-vector problem is to avoid hard "winner-take-all" updates. Parr and Russell (1995) use a differentiable approximation of the max operator and find they can produce good policies for the 4x4 and 4x3 grid problems. The approach is promising enough to warrant further study including comparisons on the difficult navigation environments described in this paper.

6 CONCLUSIONS

We can now obtain high quality policies for a class of POMDP's with nearly 100 states. We predict that these techniques can be honed to produce good policies for a wide variety of problems consisting of hundreds of states. But to handle the thousands of states needed to address realistic problems, other techniques will be needed.

Other approaches to scaling up, including various kinds of factoring and decomposition of the transitions and belief states (e.g., the sort of approach Boutilier et al. (1995) and Nicholson and Kaelbling (1994) used in fully observable domains), may be able to be used in concert with techniques described in this paper to yield practical results in moderately large POMDP problems.

References

Astrom, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Appl.*, 10:174–205.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall.

Boutilier, C., Dearden, R., and Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proceedings of the International Joint Conference on Artificial Intelligence.*

Cassandra, A. (1994). Optimal policies for partially observable Markov decision processes. Technical Report CS-94-14, Brown University, Department of Computer Science, Providence RI.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA.

Cheng, H.-T. (1988). *Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, University of British Columbia, British Columbia, Canada.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proc. Tenth National Conference on AI (AAAI).*

Connell, J. and Mahadevan, S. (1993). Rapid task learning for real robots. In *Robot Learning.* Kluwer Academic Publishers.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6).

Kushmerick, N., Hanks, S., and Weld, D. (1993). An Algorithm for Probabilistic Planning. Technical Report 93-06-03, University of Washington Department of Computer Science and Engineering. To appear in *Artificial Intelligence.*

Littman, M., Cassandra, A., and Kaelbling, L. (1995). Learning policies for partially observable environments: Scaling up. Technical Report CS-95-11, Brown University, Department of Computer Science, Providence RI.

Littman, M. L. (1994). The Witness algorithm: Solving partially observable Markov decision processes. Technical Report CS-94-40, Brown University, Department of Computer Science, Providence, RI.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research*, 28:47–66.

McCallum, R. A. (1992). First results with utile distinction memory for reinforcement learning. Technical Report 446, Dept. Comp. Sci., Univ. Rochester. See also Proceedings of Machine Learning Conference 1993.

Moore, A. W. (1994). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. In *Advances in Neural Information Processing Systems 6*, San Mateo, CA. Morgan Kaufmann.

Nicholson, A. and Kaelbling, L. P. (1994). Toward approximate planning in very large stochastic domains. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, Stanford, California.

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450.

Parr, R. and Russell, S. (1995). Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Puterman, M. L. (1994). *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY.

Ross, S. M. (1983). *Introduction to Stochastic Dynamic Programming*. Academic Press, New York.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error backpropagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 1: Foundations*, chapter 8. The MIT Press, Cambridge, MA.

Russell, S. J. and Norvig, P. (1994). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088.

Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2).

Tsitsikilis, J. N. (1994). Asynchronous stohcastic aproximation and Q-learning. *Machine Learning*, 16(3).

Watkins, C. J. (1989). *Learning with Delayed Rewards*. PhD thesis, Cambridge University.

Williams, R. J. and Baird, L. C. I. (1993). Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-13, Northeastern University, College of Computer Science, Boston, MA.