Learning policies for partially observable environments: Scaling up

Michael L. Littman mlittmancs.brown.edu

Anthony R. Cassandra arccs.brown.edu

Leslie Pack Kaelbling lpkcs.brown.edu

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

July 28, 1995

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 exist 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 hy-

brid 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.

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, $T : S \times A \to \Pi(S)$, 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 $R : S \times A \to \mathbb{R}$. The agent's decisions are made based on information from its sensors (observations) formalized by $O : S \times A \to \Pi(\Omega)$.

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.

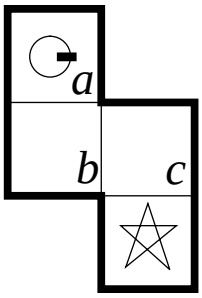

Figure 1: A tiny navigation environment.

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 $Q_{\text{MDP}}(s, a)$ 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 $Q_a(b) = \sum_s b(s)\ Q_{\text{MDP}}(s,a)$. 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.

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.

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. The relationship between these two methods is discussed in a later section of the paper.

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. Detailed descriptions of all of these problems are given in the appendix.

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.

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 which is shown in the appendix. During learning, actions were selected to maximize the current Q functions with a 0.1 probability of being

	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.

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 not much more than half a second on any problem. The learning algorithms, by contrast,

Name	Repl-Q	Lin-Q	Q_{MDP}	Optimal
Shuttle	30	30	0.10	—
Cheese Maze	46	46	0.21	1500
Part Painting	18	18	0.10	7000
4x4 Grid	75	75	0.54	7000
Tiger	12	12	0.06	7000
4x3 Grid	44	44	0.25	—

Table 3: Approximate running times for extremely small POMDP's (in seconds).

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. Table 3 summarizes the approximate running times for the various algorithms.

There are two significant exceptions to the overall performance trend shown in Table 2: 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 exploratory actions; this problem can be easily fixed by adjusting the probability of taking a random action. This combination of determinism and lack of exploration can cause the goal to be infrequently visited during learning in cases where the random initial policy leads to cyclic behavior. This was verified by stepping through the learning algorithm in those sub-optimal cases and observing the cyclic behavior.

To provide further evidence that cyclic behavior was causing the poor results, we ran the entire set of experiments a second time with an increased rate of exploration. In the results shown in Table 4 came from experiments that were identical in all parameters except the exploration probability. In these experiments there was a 0.25 probability that a random action would be chosen.

	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	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$
Linear Q	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$
optimal	—	0.186 ± 0.002	0.170 ± 0.012	0.192 ± 0.002	1.041 ± 0.180	—

Table 4: Results of POMDP solution methods using higher exploration probability.

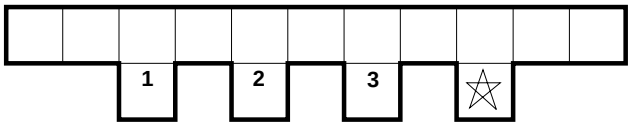

Figure 2: Navigation environment with 57 states.

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 left, turn around) and have extremely noisy transitions and observations. The appendix gives the full details of the dynamics of these environments, including transition and observation probabilities.

We ran the same collection of algorithms on these two environments with a slight change: truncated exact value iteration was given roughly 1000 seconds. This increase in time compensates for the longer running times required by the learning algorithms on these environments. Performance was measured slightly differently. The policies were evaluated for 251 trials, each consisting of a run from the problem-specific initial belief state to the goal.

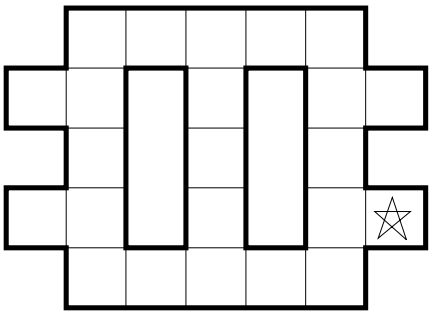

Figure 3: Navigation environment with 89 states.

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

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 5 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 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

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

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 6 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 6. 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

	57 states		89 states	
	goal%	median	goal%	median
Linear Q	99.2	14	83.7	33

Table 7: Experiments using 500,000 learning steps.

set of actions that can be chosen by the Q_{MDP} value method and reran the evaluation with results given in Table 6. Surprisingly, 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.

The results in Table 6 were for our initial experiments using 75,000 learing steps. The nature of the algorithm is such that it can improve over time, so the natural question is what happens when we run the algorithm for longer periods. We wouldn't expect the 57 state problem results to improve much, but there is considerable room for improvement on the 89 state example.

Table 7 shows the results of running the Linear-Q learning algorithm for 500,000 steps. The learning rate was the same as the original experiments, though the exploration probability was 0.2 for these experiments. These were only based upon 11 independent experiments and used the Q_{MDP} values as initial values.

Table 7 shows that allowing more learning steps leads to better performance, most importantly, it shows that learning algorithms can do better than a policy which just suppresses the "stay-in-place" action. Some preliminary experiments have shown that even better performance could be achieved with a better learning rate adjustment schedule.

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. Below, we discuss some experiments that explore why this hybrid approach does so well.

4.1 Discussion of Biasing Initial Values

Although the use of the Q_{MDP} values helps the learning algorithms significantly, we would like to understand exactly what aspect of this bias contibutes to the improvement. There are three possible explanations: the Q_{MDP} values specify a policy that gets the agent to the goal more often allowing faster learning; the Q_{MDP} values are merely in the proper neighborhood of some good final values; or it could be some complicated interaction of both of these.

Getting to the goal more frequently is an important factor in the rate at which it can learn in these environments, since the only reward received is at the goal. If the starting values are a significant distance from the good values, there might not be enough experiences or the learning rate might not be sufficient enough to bring these values into the correct range. This illustrates the complicated interaction between the initial policy, the initial values, the learning rate and the exploration rate. Thus, it is not immediately clear which factors are most significant, nor is it clear exactly how these interact with one another.

To explore this issue, we made a significant change the the experimental setup. We would now keep two sets of vectors as the simulations progressed. The first set would be the Q_{MDP} values, which we would use to determine the actions to take. The second set were randomly initialized, as in the first set of experiments, and would be the values that were actually updated. Thus, we separated the policy from the values being updated.

This setup was not the ideal, since our policy is not able to improve over time as it could in the learning experiments. If the values being updated were to ever specify a policy that was better than the policy specified by the Q_{MDP}, then continuing with the latter policy would be handicapping this set of experiments. However, running these experiments showed that the learned policy never approached the quality of the policy for the original Q_{MDP} values as shown in Table 8. For this experiment we ran only 11 independent trials on the 57 state example and used a learning rate of 0.1 for the first $50,000$ steps and 0.01 for the remaining $25,000$ steps. Since the learning rate is different from the previously presented results, we also show the results for using the random values only at this new learning rate in the table and no significant improvement results. For the separation of policy and values, the random values were within the range -20 to $+20$.

As can be seen in Table 8, the policy specified by the Q_{MDP} are not solely

	57 states	
	goal%	median
Q_{MDP}	47.4	> 251
Linear Q (Random values/policy -20 to $+20$)	15.5	> 251
Linear Q (Q_{MDP} policy and random values)	1.6	> 251

Table 8: Result of separating the policy from the updated values.

responsible for the the good results of the hybrid algorithm. To understand what role the initial values played in the hybrid algorithm, we used random initial values and policy as before, except this time we restricted the range of starting random values to be within the range of the Q_{MDP} values, $+1.0$ to $+2.5$. Using the same experimental setup we ran 8 independent experiments and found the performance to be as good as when we seeded it with the Q_{MDP} values: the goal was reached 99% of the time and there was a median of 14.5 steps to the goal. This data shows that the actual Q_{MDP} policy is of little or no help in the hybrid algorithm, and that the initial range of values is what is most important.

This result does not negate the usefulness of the hybrid algorithm, since the Q_{MDP} present a disciplined technique for finding a good range of values for initialization. However, an open question is whether the actual values are important or is it merely the relative values of the vectors. If the values all start in the same small range, then the small changes made to the values in the learning algorithm, can modify the policy significantly. It could be that any small random initial interval of values would work as well as the $+1.0$ to $+2.5$ range has. We have not yet explored this issue.

The importance of the the initial values is directly tied to the learning rate. In the original experiments (-20 to $+20$) the learning rate is not large enough to quickly bring some of the extreme values to within the useful range. Since the learning rate decays, initial values at the extreme points of the initial starting range might not have progressed much in the direction that the gradient is trying to push them. Thus, it would seem that a more liberal learning rate could potentially lead to good performance even when the initial values are far from decent final values.

In our attempt to explore this issue, we ran experiments with different learning rates with the goal of trying to get the learning algorithm to produce good results when the initial values where in the range -20 to $+20$. At our original learning rate schedule, this range of initial values produced very poor

results.

This set of experiments proved quite difficult, since even when we are only considering the learning rate, the parameter space is quite large. Recall that the learning rate is decayed in a step-wise manner. This results in the need to set the learning rate for each step and to define the intervals that each step will be in effect. The interaction between these two aspects is quite important and many different settings were tried.

Table 9 shows the performance of many settings for the learning rate schedule on the 57 state problem for 11 independent trials each. As can be seen, most of the settings did not give significantly good results. However, the last one is remarkably good compared to the original results and not that much worse than the hybrid algorithm's results. We stopped our parameter exploration with this example, since it seemed to demonstrate the point that was hypothesized.

The conclusion to draw from all of this experimental exploration is that the basic Linear-Q learning algorithm can solve modest sized POMDP problems, but that it could require a significant amount of parameter tweaking. Getting the initial values in a good range to start with makes the algorithm less sensitive to the parameter settings and the Q_{MDP} values provide a disciplined way to set these initial values.

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 approximations to the Q-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: two rooms marked with minus signs in the figure are associated with negative

learn rate	interval	57 states goal%	median
0.1	[0-50,000)	5.5	> 251
0.01	[50,000-75,000)		
1.0	[0-10,000)	23.9	> 251
0.1	[10,000-25,000)		
0.01	[25,000-50,000)		
0.001	[50,000-75,000]		
10.0	[0-1,000)	4.0	> 251
1.0	[1,000-25,000)		
0.1	[25,000-50,000)		
0.01	[50,000-75,000)		
2.0	[0-1,000)	6.8	> 251
1.0	[1,000-11,000)		
0.1	[11,000-31,000)		
0.01	[31,000-75,000)		
0.5	[0-2,500)	8.8	> 251
0.1	[2,500-15,000)		
0.05	[15,000-35,000)		
0.01	[35,000-75,000)		
0.5	[0-5,000)	13.5	> 251
0.1	[5,000-25,000)		
0.05	[25,000-40,000)		
0.001	[40,000-75,000)		
0.5	[0-25,000)	83.7	22.0
0.1	[25,000-45,000)		
0.05	[45,000-55,000)		
0.001	[55,000-75,000)		

Table 9: Results for Linear-Q algorithm with various learning rate schedules.

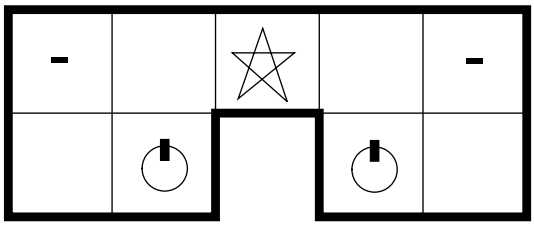

Figure 4: A 33-state navigation environment that cannot be solved with a single linear function per action.

reward, and the agent starts with equal probability facing North in one or the other of the two rooms marked with agent 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.

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 indepen-

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

dent 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 10 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.

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 Miscellaneous Issues

6.1 Sample vs. Full Backups

In the method proposed here, we use a model to generate the experiences that are used to update the values. Since we have the model, it is possible to perform full backups instead of the sample backups used in traditional Q-learning. The sample backup technique updates values based upon a single piece of experience, more specifically, based upon a single resulting state. However, when the model is known, we have access to all the possible outcomes (resulting states) and the probability of each of these occuring. With this information we can simulate many resulting states instead of just the actual resulting state.

For a POMDP the full backup is computed using all the posible resulting belief states, of which there are as many of these as number of observations. Thus the full backup version of the two learning algorithm update rules become:

$$\Delta q_a(s) = \alpha b(s)(r + \gamma \sum_{b'} P\{b'\} \max_{a'} Q^{a'}(b') - q_a(s)) \quad \text{Repl-Q},$$

$$\Delta q_a(s) = \alpha b(s)(r + \gamma \sum_{b'} P\{b'\} \max_{a'} Q^{a'}(b') - q_a \cdot b) \quad \text{Lin-Q}.$$

In these equations the sum over the resulting belief states, b', is a notaional simplification, since each resulting belief state is computed for each observation from the current belief state, b, and the current action. The probability of each resulting belief state is computed with the same machinery used to update the belief state. See [Cassandra et al., 1994] for details on updating belief states.

Although the full backup idea seems well motivated, there are some strong reasons why they were not utilized. In practice, using full backups greatly increases the running time. The increase is directly correlated with the number of observations in the problem. The full backup rule requires us to sum over all observations, making each full backup much more expensive than a sample backup. Table 11 show the timing comparisons for the sample and full backups on each of the examples shown in this paper. Notice the effect of the number of observations on the running times.

Although the increase in timing is dramatic, it isn't reason enough to rule it out as a viable method for solving these problems. In our experiments,

Name	Sample	Full	$\|\Omega\|$
Shuttle	30	115	5
Cheese Maze	46	250	7
Part Painting	18	28	2
4x4 Grid	75	270	2
Tiger	12	21	2
4x3 Grid	44	270	6
57 State	820	25500	21
89 State	1910	53300	17
33 State	280	6750	17

Table 11: Timing comparisons for full and sample backups (seconds).

though, we could discern no increase in performance for the full backup techniques using our initial experimental set-up.

Simply runnning the experiments for the same number of steps would not be sufficient to claim that full backups were no better than sample backups, since the full backups may improve the policy faster than the sample backup techniques. We explored the rate of improvement using the small problems to gauge whether or not full backups could get good policies faster than sample backups. Figures 5 through 16 shows the performance profiles for each of the small problems using two ranges for each problem. The x-axis is the number of learning updates, while the y-axis is the average reward received. These profiles were generated by periodically interrupting the learning algorithm and performing the evaluation simulation. Although there are instances where full backups seem better, there are also some where they were no better or even worse. The minor gains shown by this data, did not seem to justify the significant amount of extra computational time required to perform the full backups. This is not conclusive though, since these specific problems are not necessarily representative of the interesting problems that would actually need to be solved.

One final reason why we chose not to explore full backups further, is that it seems to contradict one of the motivations for using simulations to learn the values. We argue that the space to be explored is huge, and that using simulations will focus the search on the interesting parts of the belief space. With the introduction of full backups, we will be considering belief states we have not yet visited, since we will need to sum over all possible next belief states. This could include many belief states that are unlikely; i.e., the

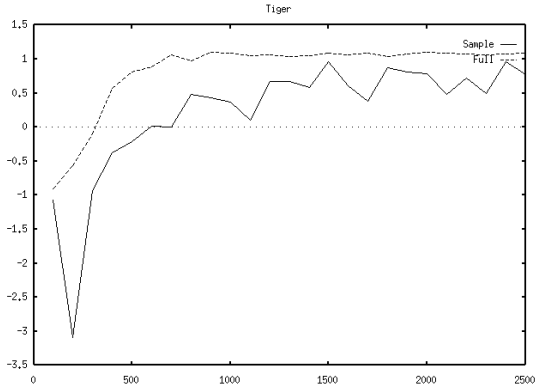

Figure 5: Performance profile of tiger problem for 2,500 steps.

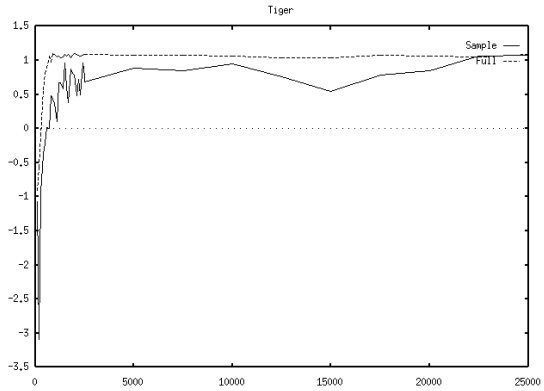

Figure 6: Performance profile of tiger problem for 25,000 steps.

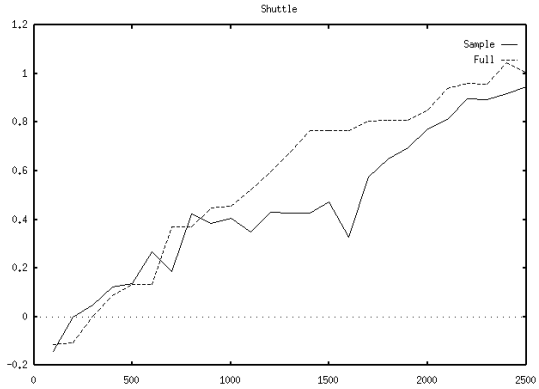

Figure 7: Performance profile of shuttle problem for 2,500 steps.

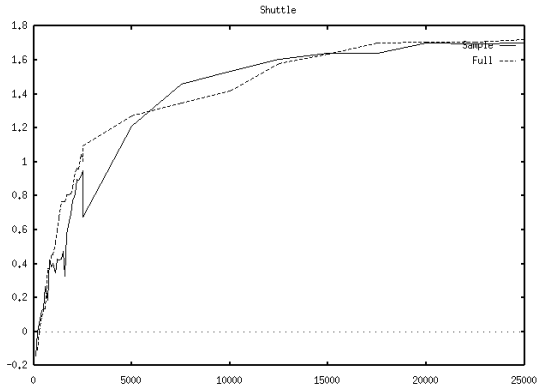

Figure 8: Performance profile of shuttle problem for 25,000 steps.

Figure 9: Performance profile of part painting problem for 2,500 steps.

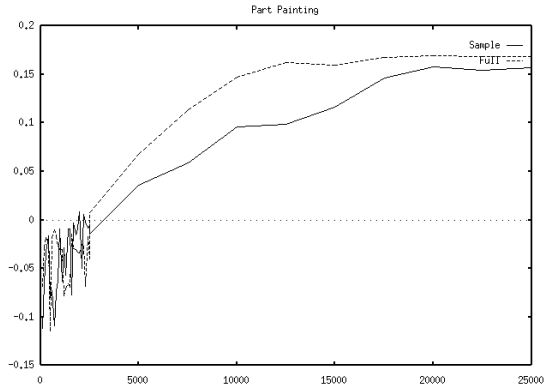

Figure 10: Performance profile of part painting problem for 25,000 steps.

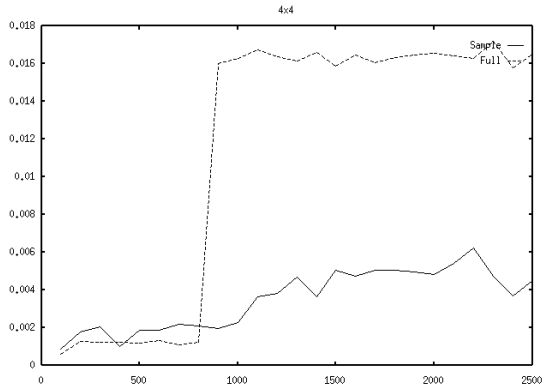

Figure 11: Performance profile of 4x4 grid problem for 2,500 steps.

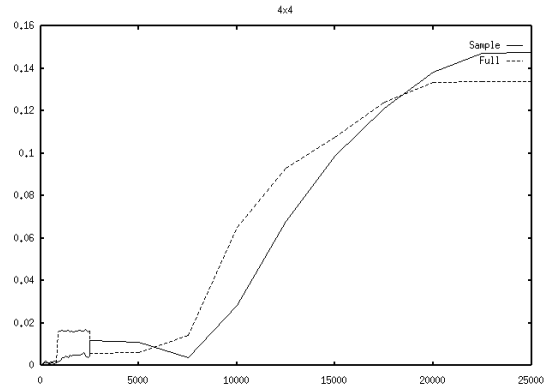

Figure 12: Performance profile of 4x4 grid problem for 25,000 steps.

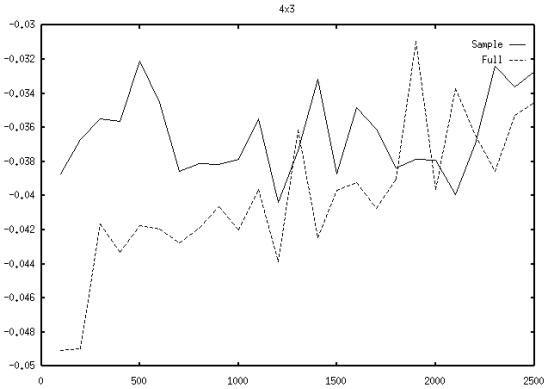

Figure 13: Performance profile of 4x3 grid problem for 2,500 steps.

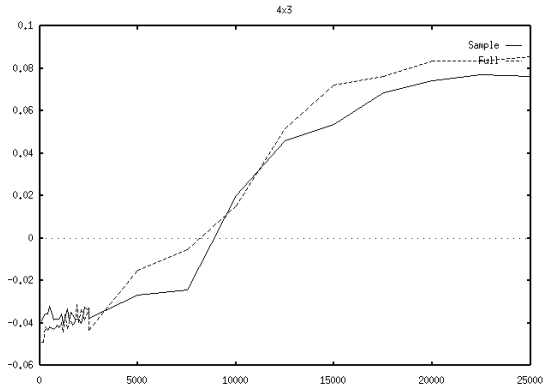

Figure 14: Performance profile of 4x3 grid problem for 25,000 steps.

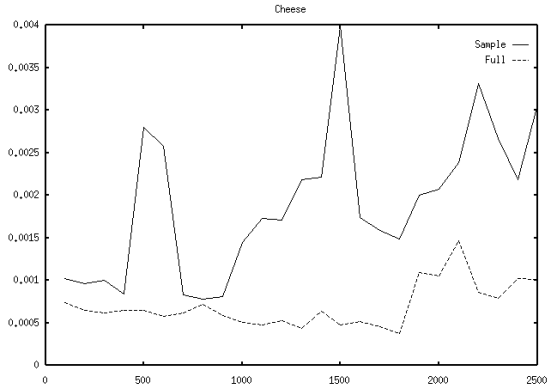

Figure 15: Performance profile of cheese maze problem for 2,500 steps.

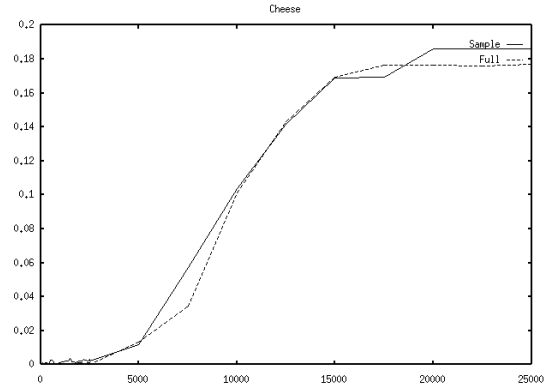

Figure 16: Performance profile of cheese maze problem for 25,000 steps.

observation that leads to that belief state has low probability. The inclusion of unlikely belief states defeats one of the motivating points in favor of using simulations.

However, a comprise approach could prove fruitful. If we do a partial backup, considering only the most likely next belief states, we might be able to keep the run times reasonable, while increasing the rate that good values are obtained. We have not explored this option, since it would require a different organization of the code to be able to efficiently find the best observations, given a particular belief state.

7 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.

A Detailed Problem Descriptions

For each of the POMDP problems presented in this report, we give a short description, its source and then a detailed description of the problem including the specific probabilities and rewards. For the extremely small problems we include the actual files used in the experiments so that we can provide as precise information as possible. Unfortunately, the format of these files requires explanation, so we must first provide the description of this file format.

For all problems, we used a discount factor of 0.95 and state-action pair values were considered to be rewards, so that higher values are more desirable. Each of the problems has a notion of a goal to be reached. When this goal is reached, the problem is restarted using a problem specific belief state as

the initial belief state. These reset belief states are given below for each of the problems.

A.1 File Format Description

Since we present the actual data files used for most of the experiments, we need to describe the format that was used for these files. We begin the desscription of the file format with some notational conventions:

- %f - a floating point number
- %d - and integer

Comma separated items within brackets, e.g., [a, b, c] represent a choice of one of these items. Everything from a '#' symbol to the end-of-line is treated as a comment. They can appear anywhere in the file. Throughout all definitions, whitespace (space, tab, newline) serves as a delimiter, with any amount of consecutive whitespace acting as a single delimiter.

The following 5 lines must appear at the beginning of the file. They may appear in any order as long as they preceed all specifications of transition probabilities, observation probabilities and rewards.

```
discount:  %f
values:  [ reward, cost ]
states:  [ %d, list of states ]
actions:  [ %d, list-of-actions ]
observations:  [ %d, list-of-observations ]
```

The definition of states, actions and/or observations can be either a number indicating how many there are or it can be a list of strings, one for each entry. These mnemonics cannot begin with a digit. For instance, both:

```
actions:  4
actions:  north south east west
```

will result in 4 actions being defined. The only difference is that in the latter the actions can then be referenced by the mnemonic name. Even when mnemonic names are used, later references can use a number as well, though

it must correspond to the positional number in this list where the numbering starts with 0. The numbers are assigned consecutively from left to right in the listing starting with zero (e.g., the action 'south' is action number 1 in the example above).

When listing states, actions or observations one or more whitespace characters are the delimiters (space, tab or newline). When a number is given instead of an enumeration, the individual elements will be referred to by consecutive integers starting at 0.

After this preamble, there is the option of specifying a problem specific starting state, or starting state distribution. If this starting state specification does not appear, then the starting belief state will be a uniform distribution over all possible states. To specify a starting state there are three formats that can be used:

```
start:  %f %f ... %f
start include:  list-of-states
start exclude:  list-of-states
```

The first form allows an explicit initial belief state definition, where the probability for each state is given. Since this can be cumbersome for large state spaces, the latter two provide more convenient specifications. The "start include" line allows you to specify a uniform distribution over only a subset of the states. The states can either be listed by number or mnemonic. The inclusion of a single state will result in certainty in the starting state. The "start exclude" mnemonic is similar, except it will define a uniform distribution over all the states that are not included in the state list that follows it.

After the initial five lines and optional start state, the speciifications of probabilities and rewards appears. These specifications may appear in any order, and can even be interleaved. Any probabilities or rewards not specified in the file are assumed to be zero.

You may also specify a particular probability or reward more than once. The definition that appears last in the file is the one that will take affect, overriding any previous declarations. This is convenient for specifying exceptions to general specifications.

To specify an individual state transition probability:

```
T: action :  start-state :  end-state :  %f
```

Anywhere an action, state or observation can appear, you can also put the wildcard character '*' which means that this is true for all possible entries that could appear here. For example:

```
T: 5 :  * :  0 1.0
```

is interpreted as action 5 always moving the system state to state 0, no matter what the starting state was (i.e., for all possible starting states.)

To specify a particular row of a transition matrix:

```
T: action :  start-state
%f %f ...  %f
```

Where there is an enter for each possible next state. This allows defining the specific transition probabilities for a particular starting state only. Instead of listing the numbers the mnemonic word `uniform` may appear. In this case, each transition for each next state will be assigned the probability $\frac{1}{N}$, with N being the number of states. Again, an asterick in either the action or start-state position will indicate all possible entries that could appear in that position.

To specify an entire transition matrix for a particular action:

```
T: action
%f %f ...  %f
%f %f ...  %f
...
%f %f ...  %f
```

Where each row corresponds to one of the start states and each column specifies one of the ending states. Each entry must be separated from the next with one or more white-space characters. The new-lines are just for formatting convenience and do not affect the final matrix results. The only restriction is there must be NxN values specified where 'N' is the number of states.

In addition, there are a few mnemonic conventions that can be used in place of the explicit matrix above:

```
identity
```

```
uniform
```

Note that `uniform` means that each row of the transition matrix will be set to a uniform distribution. The `identity` mnemonic will result in a transition matrix that leaves the underlying state unchanged for all possible starting states.

To specify individual observation probabilities:

```
O : action :  end-state :  observation %f
```

The asterick wildcard is allowed in any of the positions.

To specify a row of a particular actions' observation probability matrix:

```
O : action :  start-state
%f %f ...  %f
```

This specifies a probability of observing each possible observation for a particular action and ending state. The mnemonic short-cut `uniform` may also appear in this place.

To specify an entire observation probability matrix for an action:

```
O: action
%f %f ...  %f
%f %f ...  %f
...
%f %f ...  %f
```

The format is similiar to the transition matrices except the number of entries must be NxO where 'N' is the number of states and 'O' is the number of observations. Here too the `uniform` mnemonic can be substituted for an enire matrix. In this case it will assign each entry of each row the probability $\frac{1}{Z}$, where Z is the number of possible observations.

To specify individual rewards:

```
R: action :  start-state :  end-state :  observation %f
```

For any of the entries, an asterick for either state, action, observation indicates a wildcard that will be expanded to all existing entities.

There are two other forms to specify rewards:

```
R: action :  start-state :  end-state
%f %f ...  %f
```

This specifies a particular row of a reward matrix for a particular action and start state. The last reward specification form is

```
R: action :  start-state
%f %f ...  %f
%f %f ...  %f
...
%f %f ...  %f
```

which lets you specify an entire reward matrix for a particular action and start-state combination.

A.2 Shuttle

This problem is nearly identical to the one presented in [Chrisman, 1992], since it was obtained directly from a section of Lonnie Chrisman's code, whom was very gracious in sending it to us. It models a simple space shuttle docking problem, where we must dock by backing up into one of two space stations. The goal is to alternate between the two stations, delivering supplies. Alternatively, the goal can be view as trying to go to the station which we have least recently visited. There are penalties for bumping into the space station (trying to go forward while facing directly in front of it) and rewards for getting to the proper station (backing up into the station, which simulates docking). There is no reward or penalty for docking with the most recently visited station, it just does not receive the reward it would have if it docked with the proper, least recently visited station.

Figure 17 shows this problem pictorially. There are three actions that can be chosen: go-forward, turn-around and backup. There are eight states as depicted by the eight shuttles in Figure 17. The left-most and right-most show the shuttle states when it is docked in one of the two space stations. The most recently visited and least recently visited station are labelled with MRV and LRV respectively. It is an odd property of this problem that once the shuttle is docked in the LRV station, its next action actually appears

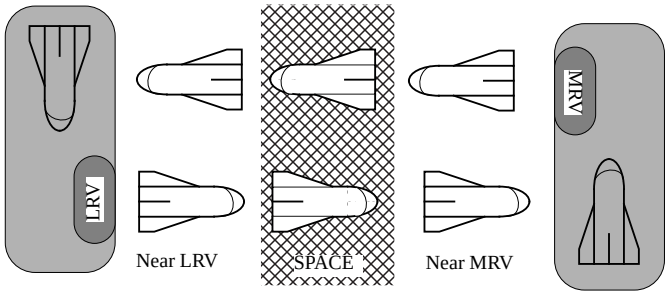

Figure 17: Chrisman's space shuttle docking problem.

to take effect from the MRV station. This results because the least recently visited station becomes the most recently visited station the instant you visit it; i.e., the LRV and MRV stations are effectively swapped. Thus, the backup action when in the LRV station transitions to the docked in MRV station state.

The penalty for bumping into the space station is -3; this occurs when the shuttle is adjacent and facing a station and chooses the go-forward action. The reward for correctly docking with the least recently visited station is $+10$; this occurs when the backup action is chosen when the suttle is adjacent, but facing away from a station.

Both the go-forward and turn-around actions are deterministic, but there is noise in the backup action. If the shuttle is adjacent and facing a space station, then the backup action: has only a 0.3 probability of succeeding; a 0.4 probability of having no effect; and has a 0.3 probability of acting like the turn-around action. If the shuttle is in a station, then the backup action will deterministically leave the shuttle in the MRV station. When the shuttle is in one of the two states out in space (shown shaded in Figure 17), then backup succeeds with probability 0.8, has no effect with probability 0.1 and with probability 0.1 has the same effect as a backup and turn-around action combination. Finally, when the shuttle is adjacent and facing away from the station, it has a 0.7 probability of actually docking with the space station and 0.3 probability of resulting in no change of state.

There are five observation for this problem which correspond to what we might see out the front window of the shuttle. We can either see one of the two space station in front of us, or we can see that we are docked in one of the two space stations, or we might not be able to see anything.

The observations correponding to being docked are always seen with prob-

ability 1.0 when the shuttle actually is docked. When the shuttle is adjacent and facing a station, it deterministically observes that station. If it is adjacent, but facing away from the station, it will deterministically see nothing. Therefore, the only interesting observation is what is observed from the space positions. When in the space states, there is a 0.7 probability that we will be able to see the station in front of us and a 0.3 probability that we see nothing. Here is the complete file specification.

```
# This is an example that appears in Lonnie Chrisman's
# paper "Reinforcement Learning with Perceptual Aliasing:
# The Perceptual Distinctions Approach", AAAI-92 The actual
# values were sent to Michael Littman from Lonnie via email
# and taken directly from Lonnie's code.

# LRV - least recently visited, MRV - most recently visited
# Backin up while docked has no effect (except to change LRV
# to MRV) Turning around while docked, leaves you in front
# of station, facing it

discount: 0.95
values: reward
states:  8
# 0  Docked in LRV
# 1  Just outside space station MRV, front of ship
#    facing station
# 2  Space facing MRV
# 3  Just outside space station LRV, back of ship
#    facing station
# 4  Just outside space station MRV, back of ship
#    facing station
# 5  Space, facing LRV
# 6  Just outside space station LRV, front of ship
#    facing station
# 7  Docked in MRV

actions: TurnAround GoForward Backup
observations: 5
```

```
# 0    See LRV forward
# 1    See MRV forward
# 2    See that we are docked in MRV
# 3    See nothing
# 4    See that we are docked in LRV

start:
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0

T: TurnAround
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0

T: GoForward
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0

T: Backup
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
0.0 0.4 0.3 0.0 0.3 0.0 0.0 0.0
0.0 0.0 0.1 0.8 0.0 0.0 0.1 0.0
0.7 0.0 0.0 0.3 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.3 0.0 0.0 0.7
0.0 0.1 0.0 0.0 0.8 0.1 0.0 0.0
0.0 0.0 0.0 0.3 0.0 0.3 0.4 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
```

```
O: *
0.0 0.0 0.0 0.0 1.0
0.0 1.0 0.0 0.0 0.0
0.0 0.7 0.0 0.3 0.0
0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 1.0 0.0
0.7 0.0 0.0 0.3 0.0
1.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0

R: GoForward : 1 : 1 : * -3
# R: GoForward : 7 : 6 : * -3  # What Chrisman specifies
R: GoForward : 6 : 6 : * -3   # What I think it should be
R: Backup : 3 : 0 : * 10
```

A.3 Cheese Maze

This problem is shown in Figure 18. There are four actions corresponding to movement in the four compass directions. Movements that attempt to move outside the grid are considered to have no effect on the location. It is as if it merely bumps into the wall and stays put. Movement is deterministic for all actions and a reward of 1.0 is given in the goal state only (zero reward for all other states.) The goal state is depicted with the star.

There are 11 states, but only 7 observations. The observations correspond to what would be seen in all four directions immediately adjacent to the location and are deterministic so that only a single observation is possible from any particular state. For example, states 5, 6 and 7 all appear identical, while states 0, 2 and 4 are all unique. The goal is its own unique observation.

```
# The infamous cheese-maze example (well.. it isn't
# much of a maze)

discount: 0.95
values: reward
states: 11
actions: N0 S0 E0 W0
```

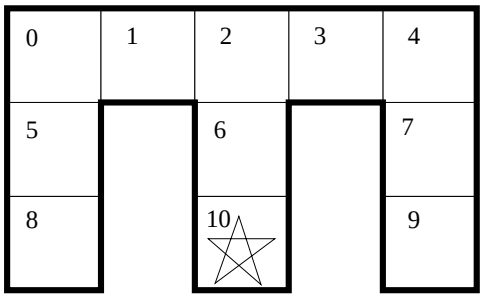

Figure 18: McCallum's cheese maze problem.

```
observations: 7

start:
0.100000 0.100000 0.100000 0.100000 0.100000
0.100000 0.100000 0.100000 0.100000 0.100000 0.0

T: N0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.100000 0.100000 0.100000 0.100000 0.100000 0.100000
 0.100000 0.100000 0.100000 0.100000 0.0

T: S0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
```

```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.100000 0.100000 0.100000 0.100000 0.100000 0.100000
 0.100000 0.100000 0.100000 0.100000 0.0

T: E0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.100000 0.100000 0.100000 0.100000 0.100000 0.100000
 0.100000 0.100000 0.100000 0.100000 0.0

T: W0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.100000 0.100000 0.100000 0.100000 0.100000 0.100000
 0.100000 0.100000 0.100000 0.100000 0.0

O: N0
1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0
```

```
0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0

O: S0
1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0

O: E0
1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0

O: W0
1.0 0.0 0.0 0.0 0.0 0.0 0.0
```

```
0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0

R: * : * : 10 : * 1.0
```

A.4 Part Painting

This problem is loosely based upon a problem formulated in [Kushmerick et al., 1993]. There is a part that needs to be painted and shipped if it is not flawed, or simply rejected if it is flawed. A flawed part starts off blemished, though painting a part can hide the blemish.

There are three state variables corresponding to whether the part is: flawed, blemished or painted. The full state space becomes the cross product of these three variables, for a total of 8 states. However, since we assume that parts start in one of only two of the states, four of these state are not needed, since it is impossible for a part to attained that state from the two possible starting states. The only two possible starting states are either the part is flawed and blemished or the part is not flawed and not blemished. In both cases, the part starts off unpainted.

There are four possible actions: inspect, paint, ship or reject the part. Painting a part that is already painted will leave its state unchanged. Painted an unblemished and non-flawed part will result in an unflawed, unblemished and painted part with probability 0.9. Thus, there is a 0.1 probability that the painting process is not successful. The same goes for painting a flawed and blemished part, except that if the painting is succesful (probability 0.9) then the part also become unblemished, with the paint hiding the blemish.

Rejecting or shipping a part will result in either a penalty or reward of 1, and the state being reset to correspond to a new part with one of the two possible starting states with equal probability.

Inspecting a part does not change its state, it merely gives information about the part (i.e., an observation). There are only two observations: blemished and unblemished. Only the inspect action will yield any information, since all other actions result in a deterministic unblemished observation (which is the same as seeing nothing in these states.) Inspecting will observe the whether the part is blemished or unblemished part successfully with probability 0.75, and with probability 0.25 result in the incorrect observation.

```
# Problem based on example in some of Hanks and
# Kushmerick's papers

#  BL - blemished, FL - flawed, PA painted

discount: 0.95
values: reward
states: NFL-NBL-NPA NFL-NBL-PA FL-NBL-PA FL-BL-NPA
actions: paint inspect ship reject
observations: NBL BL

start:
0.5 0.0 0.0 0.5

T: paint : NFL-NBL-NPA : NFL-NBL-NPA
0.1

T: paint : NFL-NBL-NPA : NFL-NBL-PA
0.9

T: paint :NFL-NBL-PA : NFL-NBL-PA
1.0

T: paint : FL-NBL-PA : FL-NBL-PA
1.0

T: paint : FL-BL-NPA : FL-NBL-PA
0.9
```

```
T: paint :FL-BL-NPA : FL-BL-NPA
0.1

T: inspect
identity

T: reject : *
0.5 0.0 0.0 0.5

T: ship : *
0.5 0.0 0.0 0.5

O: paint : * : NBL
1.0

O: inspect : NFL-NBL-NPA : NBL
0.75

O: inspect : NFL-NBL-NPA : BL
0.25

O: inspect : NFL-NBL-PA : NBL
0.75

O: inspect : NFL-NBL-PA : BL
0.25

O: inspect : FL-NBL-PA : NBL
0.75

O: inspect : FL-NBL-PA : BL
0.25

O: inspect : FL-BL-NPA : NBL
0.25

O: inspect : FL-BL-NPA : BL
0.75
```

```
O: ship : * : NBL
1.0

O: reject : * : NBL
1.0

R: ship : NFL-NBL-PA : * : *
1.0

R: reject : FL-BL-NPA : * : *
1.0

R: ship : NFL-NBL-NPA : * : *
-1.0

R: reject : NFL-NBL-NPA : * : *
-1.0

R: reject : NFL-NBL-PA : * : *
-1.0

R: ship : FL-NBL-PA : * : *
-1.0

R: ship : FL-BL-NPA : * : *
-1.0
```

A.5 4x4 Grid

This problem consists of a simple 4 by 4 grid of locations. Movement is deterministic in the four compass directions, with no change of location occurring if it tries to move out of the grid. There is a single goal state in the south-east corner location where a reward of 1 is gained. All other states give zero reward. The difficulty in this problem is that all states look exactly the same, except the goal state. Thus there are two observations: we deterministically see 'Nothing' if we are in any state besides the goal and deterministically see the goal when we are in the goal location.

```
discount: 0.95
values: reward
states: 16
actions: N0 S0 E0 W0
observations: nothing goal

start:
0.066667 0.066667 0.066667 0.066667 0.066667 0.066667 0.066667
 0.066667 0.066667 0.066667 0.066667 0.066667 0.066667
 0.066667 0.066667 0.0

T: N0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.066667 0.066667 0.066667 0.066667 0.066667 0.066667 0.066667
 0.066667 0.066667 0.066667 0.066667 0.066667 0.066667
 0.066667 0.066667 0.0

T: S0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.066667 0.066667 0.066667 0.066667 0.066667 0.066667 0.066667
 0.066667 0.066667 0.066667 0.066667 0.066667 0.066667
 0.066667 0.066667 0.0

T: E0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
0.066667 0.066667 0.066667 0.066667 0.066667 0.066667 0.066667
 0.066667 0.066667 0.066667 0.066667 0.066667 0.066667
 0.066667 0.066667 0.0

T: W0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

```
0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
0.066667 0.066667 0.066667 0.066667 0.066667 0.066667 0.066667
 0.066667 0.066667 0.066667 0.066667 0.066667 0.066667
 0.066667 0.066667 0.0

O: * : * : nothing 1.0
O: * : 15 : nothing 0.0
O: * : 15 : goal 1.0

R: * : * : 15 : * 1.0
```

A.6 Tiger

This is a simple 2 state and two observation process. The state of the world can be either that there is a tiger behind the left door or there is a tiger behind the right door. We have three possible actions: listen, open the left door or open the right door. If we open the door with the tiger behind it we get a reward of -100 (i.e., a penalty). Opening the door without the tiger gives a reward of $+10$. Listening always costs, and the reward is -1 (i.e., a cost). However, listening gives us some information about where the tiger is. In particular, listening will tell us where the tiger is correctly 85% of the time and incorrectly 15% of the time.

```
# This is the tiger problem of AAAI paper fame.

discount: 0.95
```

```
values: reward
states: tiger-left tiger-right
actions: listen open-left open-right
observations: tiger-left tiger-right

start:
0.5 0.5

T:listen
identity

T:open-left
uniform

T:open-right
uniform

O:listen
0.85 0.15
0.15 0.85

O:open-left
uniform

O:open-right
uniform

R:listen : * : * : * -1

R:open-left : tiger-left : * : * -100

R:open-left : tiger-right : * : * 10

R:open-right : tiger-left : * : * 10

R:open-right : tiger-right : * : * -100
```

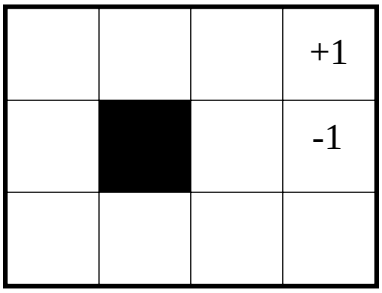

Figure 19: Russell's 4x3 maze problem.

A.7 4x3 Grid

This problem is nearly identical to that used in [Parr and Russell, 1995] and is shown in Figure 19. The only change was to remove the zero cost absorbing state and replace the transitions to it with equally likely transitions to all non-reward and non-penalty states. This has the effect of creating a reset after getting a reward or penalty.

There are 11 states in this problem as shown in Figure 19. Two of states are special states where a reward or penalty is received ($+1$ or -1) and the agent being reset afterwards. For all other states, a reward of -0.04 is incurred (i.e., a cost).

There are four actions corresponding to movements in the four major compass directions. These movements succeed with probability 0.8, and with 0.1 probability of moving perpendicular to the intended direction. Any movement that would take the agent outside the grid, results in the agent staying in the start location. For example, an attempt to move north from the extreme north-west location will result in a 0.9 probability that the agent stays in that location. This occurs because the north movement, probability 0.8, and the west movement, probability 0.1 are not possible. There is still a 0.1 probabilty that the agent will move east.

There are six observations: four that correspond to whether or not there is a wall directly to the east and west of the location and one each for the positive reward, $+1$, state and the negative reward, -1 state. All of these observations are deterministic.

```
# Russell and Norvig's 4x3 maze
```

```
# Rewards and penalties are associated with states, not actions.
# The default reward/penalty is -0.04.
# There is no discounting, but a there is an absorbing state that
# + and - transition to automatically.  The absorbing state cannot be exited.
#
# States are numbered from left to right:
#
# 0  1  2  3
# 4     5  6
# 7  8  9  10
#
# I removed the absorbing state
#
# The actions, NSEW, have the expected result 80% of the time, and
# transition in a direction perpendicular to the intended on with a 10%
# probability for each direction.  Movement into a wall returns the agent
# to its original state.
#
# Observation is limited to two wall detectors that can detect when a
# a wall is to the left or right.  This gives the following possible
# observations:
#
# left, right, neither, both, good, bad, and absorb
#
# good = +1 reward, bad = -1 penalty,

discount: 0.95
values: reward
states: 11
actions: n s e w
observations: left right neither both good bad

start:
0.111111 0.111111 0.111111 0.0 0.111111 0.111111 0.0
 0.111112 0.111111 0.111111 0.111111

T: n
0.9 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

```
0.1 0.8 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.1 0.8 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.111111 0.111111 0.111111 0.0 0.111111 0.111111 0.0
 0.111112 0.111111 0.111111 0.111111
0.8 0.0 0.0 0.0 0.2 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.8 0.0 0.0 0.1 0.1 0.0 0.0 0.0 0.0
0.111111 0.111111 0.111111 0.0 0.111111 0.111111 0.0
 0.111112 0.111111 0.111111 0.111111
0.0 0.0 0.0 0.0 0.8 0.0 0.0 0.1 0.1 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.8 0.1 0.0
0.0 0.0 0.0 0.0 0.0 0.8 0.0 0.0 0.1 0.0 0.1
0.0 0.0 0.0 0.0 0.0 0.0 0.8 0.0 0.0 0.1 0.1

T: s
0.1 0.1 0.0 0.0 0.8 0.0 0.0 0.0 0.0 0.0 0.0
0.1 0.8 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.1 0.0 0.1 0.0 0.8 0.0 0.0 0.0 0.0 0.0
0.111111 0.111111 0.111111 0.0 0.111111 0.111111 0.0
 0.111112 0.111111 0.111111 0.111111
0.0 0.0 0.0 0.0 0.2 0.0 0.0 0.8 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.1 0.1 0.0 0.0 0.8 0.0
0.111111 0.111111 0.111111 0.0 0.111111 0.111111 0.0
 0.111112 0.111111 0.111111 0.111111
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.9 0.1 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.8 0.1 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.8 0.1
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.9

T: e
0.1 0.8 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.2 0.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.1 0.8 0.0 0.1 0.0 0.0 0.0 0.0 0.0
0.111111 0.111111 0.111111 0.0 0.111111 0.111111 0.0
 0.111112 0.111111 0.111111 0.111111
0.1 0.0 0.0 0.0 0.8 0.0 0.0 0.1 0.0 0.0 0.0
0.0 0.0 0.1 0.0 0.0 0.0 0.8 0.0 0.0 0.1 0.0
0.111111 0.111111 0.111111 0.0 0.111111 0.111111 0.0
 0.111112 0.111111 0.111111 0.111111
```

```
0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.1 0.8 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.2 0.8 0.0
0.0 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.1 0.0 0.8
0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.9

T: w
0.9 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.0 0.0
0.8 0.2 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.8 0.1 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.0
0.111111 0.111111 0.111111 0.0 0.111111 0.111111 0.0
 0.111112 0.111111 0.111111 0.111111
0.1 0.0 0.0 0.0 0.8 0.0 0.0 0.1 0.0 0.0 0.0
0.0 0.0 0.1 0.0 0.0 0.8 0.0 0.0 0.0 0.1 0.0
0.111111 0.111111 0.111111 0.0 0.111111 0.111111 0.0
 0.111112 0.111111 0.111111 0.111111
0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.9 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.8 0.2 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.8 0.1 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.8 0.1

O: *
1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0
1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0
1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0

R: * : 0 : * : * -0.04
R: * : 1 : * : * -0.04
R: * : 2 : * : * -0.04
R: * : 3 : * : * 1.0
R: * : 4 : * : * -0.04
```

```
R: * : 5 : * : * -0.04
R: * : 6 : * : * -1.0
R: * : 7 : * : * -0.04
R: * : 8 : * : * -0.04
R: * : 9 : * : * -0.04
R: * : 10 : * : * -0.04
```

A.8 Larger Problems

The three larger problems (57, 89 and 33 states) are all based upon the same framework and so we will describe this first. We will then discuss any deviations from this basic scheme in the individual sections for each of those problems.

The basic idea behind these problems, is that there is an agent wandering around some office building. We assume that the locations have been discretized so that there are a finite number of locations where the agent could be. The agent has a small finite set of actions it can take, but these only succeed with some probability. Additionally, the agent is equipped with very short range sensors to provide it only with information about whether it is adjacent to a wall. These sensors also have the property that they are somewhat unreliable and will sometime miss walls or see walls when there are none. It can "see" in four directions: forward, backward, left and right. It is important to note that these observations are relative to the current orientation of the agent. Unlike most of the previous problems where the locations corresponded to states, in these problems the location *and* the agent's current orientation comprise the states.

There is a single goal location, which is the objective to be achieved by the agent. Since achieving the goal is the main concern, the goal location actually requires only a single state, since we are not interested in the orientation when it achieves the goal. Upon reaching the goal, the agent is reset randomly among the non-goal states.

The location layouts are show in Figures 2, 3 and 4, but the actual states consist of 4 for each location, since this is the number of possible orientations we allow the agent to have.

The actions that can be chosen consists of the movements: forward, turn-left, turn-right, turn-around and no-op (stay-in-place). These action are

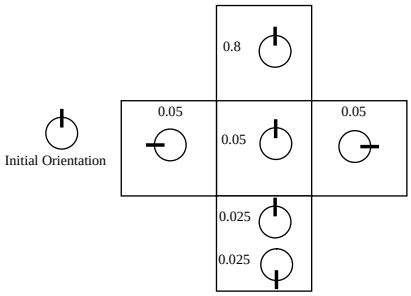

Figure 20: Transition probabilities for the forward movement in larger problems.

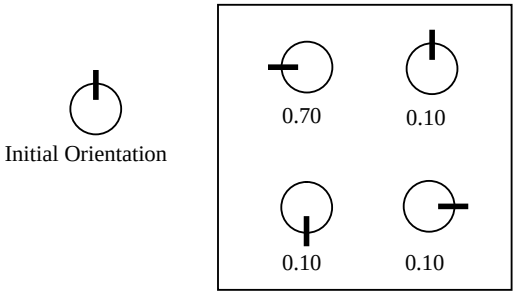

Figure 21: Transition probabilities for the turn-left movement in larger problems.

relative to the current orientation of the agent. As mentioned, each of the actions is probabilistic and are shown in Figures 20 through 23. The no-op action always succeeds in leaving the state unchanged.

There are many states, where moving forward could be impossible, given the layout of the locations. In these cases, the probability mass for the impossible next state, is collapsed into the probability of not changing state. The same goes for all other possible movement directions. For example, consider the situation where the agent has a wall to the front and the left of it. Moving forward will result in no change of state with probability 0.9. This results because the successful forward movement, probability 0.8, is impossible as is the erroneous left movement, which has probability 0.05. Thus these two probabilities are added to the predefined probability, 0.05, of having no state change. There will still be a 0.05 probability of sliding erroneously to the right and 0.25 probability for the two accidental backward movements.

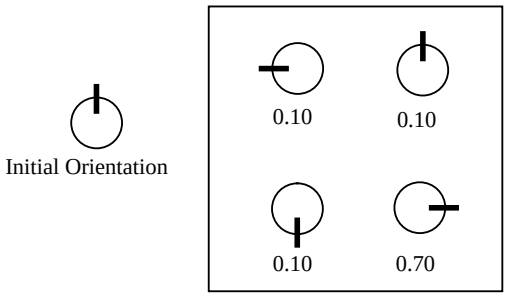

Figure 22: Transition probabilities for the turn-right movement in larger problems.

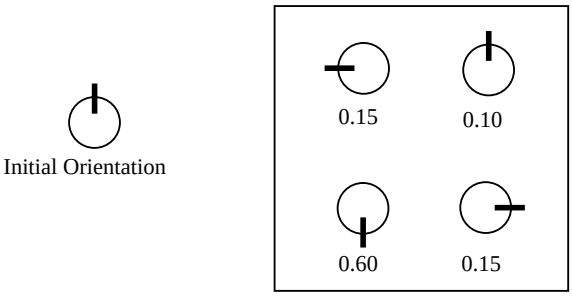

Figure 23: Transition probabilities for the turn-around movement in larger problems.

The observations are comprised of four independent sensors which are able to detect walls immediately (i.e., the adjacent locations only) in four directions. Each is not completely reliable, so that if there is a wall, it will detect it with probability 0.9. If there is no wall, then it will mistakenly sense a wall with probability 0.05. Thus, from a given state, there are 16 possible observations: 2 possible values, wall or no-wall, for 4 separate sensors. These sensors operate independently, so the probability for an observation is computed by multiplying the four individual probabilities.

The goal state gives a reward of +1 and all other states give zero reward. In addition, there is an extra observation, that corresponds to observing the goal. This observation is only possible in the goal state and occurs deterministically.

A.8.1 57 States

To the basic structure, this problem adds three more observations. These observations correspond to three distinct landmarks in the environment. There are three states where the agent will deterministically see one of these landmarks, which will fully disambiguate its location. The locations are indicated in Figure 2 and the landmark observations are made only when the agent is facing south in those locations. These landmarks are a remaining artifact from exploration of some other ideas.

A.8.2 89 States

This problem conforms exactly to the description layed out previously, no variation on rewards or observations has been made.

A.8.3 33 States

This problem deviates from the initial set-up it two ways. First, the starting belief state and the belief state it resets to is not uniform over all non-goal states. Instead it is uniform over two particular state as depicted in Figure 4. The other change is in the addition of penalties. -1.0, for the two upper corner states. These modification where made to bring out a certain characteristic of the problem.

References

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

[Bertsekas, 1987] Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, N.J.

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

[Cassandra, 1994] 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 et al., 1994] 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, 1988] Cheng, H.-T. (1988). *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, British Columbia, Canada.

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

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

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

[Kushmerick et al., 1993] 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, 1994] 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, 1991] Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research*, 28:47–66.

[McCallum, 1992] 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, 1994] 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 and Kaelbling, 1994] 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 and Tsitsiklis, 1987] Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450.

[Parr and Russell, 1995] 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, 1994] Puterman, M. L. (1994). *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY.

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

[Rumelhart et al., 1986] 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.

[Smallwood and Sondik, 1973] 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, 1978] Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2).

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

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

[Williams and Baird, 1993] 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.