Markov decision processes with observation costs

Eric A. Hansen
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
hansen@cs.umass.edu

CMPSCI Technical Report 97-01

Abstract

A partially observable Markov decision process (POMDP) is a generalization of a Markov decision process in which observation of the process state can be imperfect and/or costly. Although it provides an elegant model for control and planning problems that include information-gathering actions, the best current algorithms for POMDPs are computationally infeasible for all but small problems. One approach to this dilemma is to identify subsets of POMDPs that can be solved more efficiently than the general problem can be.

This report describes a policy iteration algorithm that we prove converges to an optimal policy for any infinite-horizon POMDP for which it is optimal to acquire perfect information at finite intervals. For this subset of POMDPs, the value function can be represented by a value for each state of the Markov process – the same representation used for completely observable MDPs – and this simplification makes it possible to compute optimal policies efficiently for many problems in this class. The policy iteration algorithm we describe is a synthesis of ideas from dynamic programming and heuristic search.

1 Introduction

Many problems of practical importance in artificial intelligence, operations research, and other decision sciences, can be modeled as a Markov decision process (MDP). In this model, a decision maker controls a stochastic process by taking a sequence of actions, with the action taken at each step determined by the current state of the process. In a standard MDP, two simplifying assumptions are made about the decision maker's awareness of the current state of the process. The first is that the decision maker observes the process before each action it takes. The second is that an observation reliably reveals the current state. Because a decision maker always knows the state of the MDP when it acts, such MDPs are said to be *completely observable.*

There are many real-world problems, however, in which one or both of these assumptions cannot be made. Observations may be too costly to make before every action, or they may provide imperfect or incomplete information. Such problems can be formalized by an extension of the MDP model called a *partially observable* Markov decision process (POMDP). (See Monahan, 1982, and Lovejoy, 1991, for surveys.) In a POMDP, the decision maker's uncertainty about the state of the process is represented by a probability distribution over the possible states that is updated by Bayes' rule after each action and observation.

POMDPs can be solved using dynamic programming to find an optimal value function and policy, where both are defined for all possible probability distributions over the states of the Markov process. This requires a complex representation of the value function and policy, however, and algorithms for solving POMDPs are computationally prohibitive for all but small problems. This paper begins with the observation that for a significant class of POMDPs, an optimal value function and policy can be represented more simply. This is the class of POMDPs for which it is optimal to take an action that provides perfect information at finite intervals. For this class of POMDPs, we show that the value function can be represented by a value for each state of the Markov process, the same representation used for completely observable MDPs, and the policy can be represented in an equally simple manner. We develop a generalization of Howard's policy iteration for this class of POMDPs, prove that it converges to an optimal policy after a finite number of iterations, and provide empirical evidence that it can solve larger problems in this class of POMDPs than current algorithms.

The class of POMDPs that can be solved with this algorithm includes problems for which observations reliably reveal the current state of the process, but incur a cost that makes it prohibitive to observe the state before each action. We call such problems "MDPs with observation costs." It also includes problems for which the action that provides perfect information is not an observation per se, but instead a *reset* that moves the process to a specific state with probability 1.0. (A mixture of a reset and an observation is an action that has some probabilistic effect on the state, then provides perfect information about the result. Such an action might be used to model an observation for which the cost is a potential change of state.)

This class of POMDPs includes several problems of practical significance. In particular, machine maintenance, quality control, and similar problems often assume an inspection

1. *Initialization*: Start with an initial policy δ.
2. *Policy Evaluation*: Compute the value function V^δ for policy δ by solving the following system of $|S|$ equations in $|S|$ unknowns given by equation (1).
3. *Policy Improvement*: For each state $s \in S$, if there is some action $a \in A$ such that $[r(s,a) + \beta \sum_{s' \in S} p(s,a,s')V^\delta(s')] > V^\delta(s)$ then $\delta'(s) = a$; otherwise $\delta'(s) = \delta(s)$.
4. *Convergence test*: If δ' is the same as δ, then the algorithm has converged to an optimal policy. Otherwise, set $\delta = \delta'$ and go to step 2.

Figure 1: Howard's policy iteration algorithm for discounted infinite-horizon MDPs

action that provides perfect information and/or a repair/replace action that resets a process to a known state (e.g., Klein, 1962; Luss, 1976; Rosenfield, 1976; Ross, 1971; Tijms & Van Der Duyn Schouten, 1984; White, 1978). An infinite-horizon version of the machine maintenance problem described by Smallwood and Sondik (1973), for example, can easily be solved by this approach. For stochastic control problems where acquiring perfect state information incurs a cost, this approach finds a policy that combines open-loop and closed-loop control. Other attempts to formalize the problem of when to acquire costly sensory feedback have assumed a fixed sensing interval (e.g., Sheridan, 1970; Abramson, 1993; Langley, Iba, and Shrager, 1994). Formalization of this problem as a POMDP shows that a state-dependent sensing interval can be more cost-effective (Hansen, 1994).

The paper is organized as follows. Section 2 reviews the MDP and POMDP models. Section 3 describes an extension of Howard's policy iteration algorithm for solving POMDPs in the class we have identified, and proves that it converges to an optimal policy after a finite number of iterations. Section 4 describes the performance of the algorithm on an illustrative example and discusses some factors that influence its efficiency.

2 Model

Consider a discrete-time MDP with a finite state set S and a finite action set A. A state transition function specifies, for each state $s \in A$ and action $a \in A$, a discrete probability distribution over resulting states $s' \in S$. Let $p(s,a,s')$ denote the probability that taking action a in state s results in state s'. A reward function specifies the expected reward for taking action a in state s, denoted $r(s,a)$.

For infinite-horizon MDPs, it is well-known that a stationary policy is optimal. If an MDP is completely observable, a stationary policy can be specified by a function δ that maps each state s to some action a. A value function, V^δ, gives the expected discounted return for following policy δ over an infinite horizon, starting from each state. It can be computed by solving the following system of $|S|$ equations in $|S|$ unknowns,

$$V^\delta(s) = r(s,\delta(s)) + \beta \sum_{s'} p(s,\delta(s),s')V^\delta(s'), \quad (1)$$

where β, with $0 < \beta < 1$, is a discount factor that ensures the infinite-horizon return is finite. An optimal value function, V^*, satisfies the Bellman equation,

$$V^*(s) = \max_a \left[r(s,a) + \beta \sum_{s'} p(s,a,s') V^*(s') \right],$$

and an optimal policy, δ^*, is a greedy policy with respect to the optimal value function. Howard's policy iteration algorithm is a well-known approach to solving infinite-horizon MDPs (Howard, 1960). Like other dynamic programming algorithms, it relies on an operation called a *backup* that is performed each iteration, for every state, until convergence. (See Figure 1.)

In a completely observable MDP, all actions provide perfect information about the state of the Markov process. In a partially observable MDP, actions may provide perfect information, imperfect information, or no information. Information is recieved in the form of an observation (or signal) from a finite set Z. An observation function specifies, for each action and resulting state, a discrete probability distribution over possible observations. Let $q(s', a, z)$ denote the probability of observing z after taking action a resulting in state s'. Because different actions provide different amounts of information, the best action in a situation depends not only on the probable effect of the action on the state, but on the information it provides the decision maker. This allows POMDPs to model more realistic problems than can be modeled by completely observable MDPs, although it also makes them more difficult to solve.

When the state of an MDP is only imperfectly or incompletely observed, an estimate of the state can be represented by a probability distribution over S. Called an *information state* (or information vector), this estimate is updated after each action and observation using Bayes' rule. Let π denote an information state and let $\pi(s)$ denote the probability that the current state of the Markov process is s. If action a is taken in information state π, and if z is subsequently observed, then a new information state π' is determined by the transition function $T(\pi|a,z)$, where by Bayes' rule,

$$\pi'(s') = T_{s'}(\pi|a,z) = \frac{\sum_s \pi(s) p(s,a,s') q(s',a,z)}{\sum_s \sum_{s'} \pi(s) p(s,a,s') q(s',a,z)}$$

Because an information state is a sufficient statistic, we can define a more general MDP in which the state set consists of the set of all possible information states, denoted $\Pi(S)$, the action set remains A, the transition function is T, and the reward function is,

$$R(\pi, a) = \sum_{s \in S} \pi(s) r(s,a).$$

This MDP over the set of all possible information states has an uncountably infinite number of states, however, and dynamic programming algorithms for finite state MDPs cannot be used to solve it. Special-purpose dynamic programming algorithms have been developed for POMDPs, but they are complex and computationally intensive and we do not describe them here. They rely on a more complex representation of the value function that

is defined for all possible information states. In this paper, we define the value function for the finite set of states in S only. Nevertheless, we show that there is a class of POMDPs that can be solved using this simpler representation of the value function.

3 Algorithm

We now describe a policy iteration algorithm that finds an optimal policy for any POMDP for which it is optimal to acquire perfect information at finite intervals. We first describe a version of the algorithm for POMDPs that have two kinds of actions: actions that provide perfect information and actions that provide no information. We then briefly describe how to extend the algorithm to solve POMDPs that also include actions that provide imperfect information.

3.1 Multi-step backup

We assume the value function is defined for the states of the underlying Markov process only, just as if the problem were completely observable. If actions provide no information (or imperfect information), this leaves the value function undefined for the terminal states of a traditional (single-step) dynamic programming backup. To get around this problem, we introduce the concept of a *multi-step backup*. This generalizes the familiar dynamic programming backup to sequences of actions. With the restriction that the last action in the sequence provides perfect information, we can ensure the value function is defined for the terminal states of the backup.

Let $a_1..a_k$ denote a k-length sequence of actions and let $p(s, a_1..a_k, s')$ denote the probability that taking actions $a_1..a_k$ from state s results in state s'. (These probabilities are easily computed from the "single-step" transition probabilities.)

For infinite-horizon problems, the number of possible action sequences that can be backed-up is infinite unless some bound is placed on the interval between actions that provide perfect information. Let A_b denote the set of all action sequences of length no greater than b, where the last action in the sequence provides perfect information and the earlier actions do not. We define a multi-step backup for state s as follows:

$$V(s) = \max_{a_1..a_k \in A_b} \left[r(s, a_1) + \sum_{i=1}^{k-1} \beta^i p(s, a_1..a_i, s') r(s', a_{i+1}) + \beta^k \sum_{s'} p(s, a_1..a_k, s') V(s') \right].$$

We call a value function created by performing multi-step backups a *multi-step value function*, and we call the corresponding policy a *multi-step policy*. It is easy to see that by using multi-step backups, policy iteration converges (in principle) to a policy that is optimal among all policies that have a bound b on the interval between actions that provide perfect information. We call this algorithm *multi-step policy iteration*.

3.2 Heuristic multi-step backup

Our definition of a multi-step backup introduces two difficulties. First, the combinatorial explosion of action sequences that can be backed-up makes exhaustive search for the best action sequence for each state computationally prohibitive. Second, an arbitrary bound b on the search depth is required to keep the search finite for infinite-horizon problems. But for any b, an optimal policy may require taking more than $b-1$ actions that provide no information before taking an action that provides perfect information. We address both of these difficulties by showing how to use a heuristic evaluation function to guide and prune the search for the best action sequence to back up.

Note that for each state s, a search tree of information states can be generated in which each information state in the tree corresponds to some action sequence that can be taken from state s. As a convenient shorthand, let $sa_1..a_k$ denote the information state reached by taking action sequence $a_1..a_k$ from starting state s. Let $f(sa_1..a_k)$ denote the backed-up value for taking action sequence $a_1..a_k$ from state s. A multi-step backup for state s is then defined as:

$$V(s) = \max_{a_1..a_k \in A_b} f(sa_1..a_k).$$

Adopting the conventional notation of heuristic search, let

$$f(sa_1..a_k) = g(sa_1..a_k) + h(sa_1..a_k),$$

where $g(sa_1..a_k)$ represents the expected reward for taking a path from state s to information state $sa_1..a_k$,

$$g(sa_1..a_k) = r(s,a_1) + \sum_{i=1}^{k-1} \beta^i \sum_{s'} p(s, a_1..a_i, s') r(s', a_i),$$

and $h(sa_1..a_k)$ represents the expected future value (after k time steps) of information state $sa_1..a_k$,

$$h(sa_1..a_k) = \beta^k \sum_{s'} p(s, a_1..a_k, s') V(s').$$

Of course, this assumes action a_k results in perfect information. If it does not, we regard h as a heuristic evaluation function that estimates the value of reaching information state $sa_1..a_i$. Similarly, we regard f as a heuristic estimate of the maximum backed-up value that can be achieved by starting in state s, taking actions $a_1..a_i$, and then taking some further actions before obtaining perfect information. We use f and h to refer to both heuristic estimates and actual values, and make the distinction between the two based on whether an action sequence ends with an action that provides perfect information or not.

Because multi-step backups create a combinatorial search problem in which the search is for the best action sequence to backup, a heuristic evaluation function can make this search more efficient. Two possible search algorithms that use a heuristic evaluation function – best-first search (also called $A*$) and depth-first branch-and-bound search – are summarized in figures 2 and 3.

The heuristic evaluation function we use assumes perfect information after actions that may not provide perfect information. As a result, f estimates the expected value for taking

1. *Set-up*: Create a search node for each action $a_i \in A$, compute $f(sa_i)$, and push the search nodes onto an OPEN list sorted in decreasing order of f.

2. *Get the next action sequence*: Remove the most promising node from OPEN and let $a_1..a_i$ be the action sequence it specifies.

3. *Possibly exit with solution*: If $a_1..a_i$ ends with an action that provides perfect information, then exit. If $f(sa_1..a_i) > V(s)$, then change the policy by setting $\delta(s) = a_1..a_i$.

4. *Possibly expand search node*: If the length of $a_1..a_i$ is less than the length bound b, then create the successor nodes of $a_1..a_i$ by adding each possible action a_j to the end of the sequence. Calculate $f(sa_1..a_ia_j)$ for each new search node, push them onto the OPEN list, and re-sort the OPEN list in decreasing order of f. Go to 2.

Figure 2: Heuristic multi-step backup for state s using best-first search.

action sequence $a_1..a_i$ plus the value of perfect information. Accordingly, we call h a *value of perfect information heuristic* (or alternatively, we refer to it as pruning based on the estimated value of perfect information). Because the value of information is always non-negative, f appears to provide an easily computed upper bound on the expected value for taking action sequence $a_1..a_i$.

If a heuristic evaluation function returns, for every state, an upper bound on the backed-up value of the best solution that passes through that state, it is said to be *admissible*. Formally, h is admissible if $h(sa_1..a_i) \geq h(sa_1..a_i..a_k)$ for every information state $sa_1..a_i..a_k$ where action a_k provides perfect information. When h is admissible, the backed-up value of a heuristic multi-step backup is guaranteed to be optimal, that is, it is guaranteed to be the same as the backed-up value of a multi-step backup that relies on exhaustive search. However, our value of perfect information heuristic is *not* admissible because it estimates the value of perfect information based on the current value function, which is a lower – not an upper – bound on the optimal value function. Therefore special justification is needed for using this heuristic in our policy iteration algorithm.

3.3 Policy iteration with heuristic multi-step backups

Because the current value function is not an upper bound on the optimal value function, it is possible for the heuristic value of an information state to underestimate the value of the best action sequence that passes through it. In spite of this, we prove that use of this heuristic evaluation function guarantees that policy improvement with heuristic multi-step backups improves a policy that is suboptimal. We show this by showing that while policy improvement with heuristic multi-step backups may not find the best policy based on the current value function, using this inadmissible heuristic, it always finds an improved policy when the current policy is not optimal. For an iterative algorithm like policy iteration, this is sufficient to ensure eventual convergence to an optimal policy.

1. *Set-up*: Create a search node for each action $a_i \in A$, compute $f(sa_i)$, and push the search nodes onto a stack.

2. While the search stack is not empty, do the following:

   (a) *Get the next action sequence*: Pop a search node off the stack and let $a_1..a_i$ be the action sequence it specifies, and let $f(sa_1..a_i)$ be its estimated value.

   (b) *Possibly update policy*: If $a_1..a_i$ ends with an action that provides perfect information and if $f(sa_1..a_i) > V(s)$, then let $\delta(s) = a_1..a_i$ and $V(s) = f(sa_1..a_i)$.

   (c) *Possibly expand search node*: If $a_1..a_i$ does not end with an action that provides perfect information, if $f(sa_1..a_i) > V(s)$, and if the length of $a_1..a_i$ is less than the length bound b, then create the successor nodes of $a_1..a_i$ by adding each possible action a_j to the end of the sequence. Calculate $f(sa_1..a_ia_j)$ for each new search node and push them onto the stack.

Figure 3: Heuristic multi-step backup for state s using depth-first branch and bound search

Theorem 1 *If heuristic multi-step backups are used in the policy improvement step, an improved policy is found whenever the current policy is not optimal.*

Proof. We prove this by showing that the policy found by policy improvement has a backed-up value for every state that is as great or greater than the value of that state under the current policy – and strictly greater for at least one state whenever the current policy is suboptimal. (The backed-up value is the value returned by a multi-step backup.) By Howard's proof of the convergence of policy iteration, it follows that the value of every state under the new policy (after policy evaluation) must be as great or greater than its value under the current policy – and greater for at least one state.

Recall that the action sequence specified for a state is not changed unless an action sequence with a greater backed-up value is found by performing a heuristic multi-step backup. It is immediate that the backed-up value for every state after policy improvement is greater than or equal to the current state value. To show that an improved policy is found whenever the current policy is suboptimal, all we must show is that for at least one state a new action sequence is found with a backed-up value that is greater than the value of that state under the current policy. This is proved by induction on the length of action sequences.

If the value of any state can be improved by taking a single action that provides perfect information, then an improved policy is found because each action sequence of length one is evaluated. Now, make the inductive hypothesis that an improved policy is found if the value of any state can be improved by taking an action sequence of length less than k. We show that an improved policy must be found if the value of any state can be improved by taking an action sequence of length k. Suppose there is a length k action sequence $a_1..a_k$ such that $f(sa_1..a_k) > V(s)$. If $a_1..a_k$ is evaluated in the course of the search, an improved policy is found and the theorem holds. If it is not, let $a_1..a_i$ be the action sequence at which the branch of the search tree leading to $a_1..a_k$ was pruned. Because the search tree

is pruned at this node, we have $V(s) \geq f(sa_1..a_i)$. By assumption, $f(sa_1..a_k) > V(s)$, and so,

$$f(sa_1..a_k) > f(sa_1..a_i).$$

Expanding this inequality we get

$$g(sa_1..a_i) + \beta^i \sum_{s'} p(s, a_1..a_i, s') \left[g(s' a_{i+1}..a_k) + \beta^{k-i} \sum_{s''} p(s', a_{i+1}..a_k, s'') V(s'') \right] > g(sa_1..a_i) + \beta^i \sum_{s'} p(s, a_1..a_i, s') V(s'),$$

and simplification yields

$$\sum_{s} p(s, a_1..a_i, s') \left[g(s' a_{i+1}..a_k) + \beta^{k-i} \sum_{s''} p(s', a_{i+1}..a_k, s'') V(s'') \right] > \sum_{s'} p(s, a_1..a_i, s') V(s').$$

Therefore, there is some state, s', such that $f(s' a_{i+1}..a_k) > V(s')$.

In other words, if for some state s there is an action sequence $a_1..a_k$ with a greater value than the value of the current policy for state s, and if the branch leading to $a_1..a_k$ was pruned at action squence $a_1..a_i$, then there must be some other state, s', for which action sequence $a_{i+1}..a_k$ has a greater value than $V(s')$. But action sequence $a_{i+1}..a_k$ has length less than k. By the inductive hypothesis, it follows that policy improvement finds an improved policy. □

Although this search heuristic resembles other pruning techniques for accelerating dynamic programming, such as temporary action elimination (Puterman & Shin, 1982), it differs in several respects. Most significantly, it uses the current value function for pruning, which is neither an upper bound on the optimal value function nor an upper bound on the improvement of the value function after one more iteration. By contrast, action elimination uses an upper bound for pruning. Moreover, the upper bound it uses can be calculated for any problem solvable by dynamic programming, whereas the estimated value of perfect information heuristic we use for pruning is only applicable to POMDPs. Finally, action elimination prunes actions to accelerate single-step backups. Our approach is to prune states that result from sequences of actions.

3.4 Convergence

Bounding the number of actions providing no information that can be taken before an action providing perfect information ensures the policy improvement step terminates each iteration. It also keeps the number of possible policies finite. Because policy improvement monotonically improves a sub-optimal policy, it follows that policy iteration converges after a finite number of iterations to a policy that is optimal up to this bound.

A question that remains is whether policy iteration can find a policy that is optimal without any bound on the interval between actions that provide perfect information. If

policy iteration converges and the length bound is used to prune any search path during the last iteration, it is possible the policy to which it converges could be improved by allowing longer action sequences before acquiring perfect information. If policy iteration converges without invoking the length bound to prune any search path during the last iteration, however, the policy to which it converges must be optimal without any bound on the interval between actions that provide perfect information. This follows from Theorem 1, and it suggests a modification of the standard convergence test for policy iteration. If policy iteration converges by the standard criterion (that is, the policy is the same from one iteration to the next) but uses the length bound to prune some search path during the last iteration, the algorithm is continued for one more iteration with an increased length bound. Two conditions must now be satisfied for convergence.

Modified convergence test:

1. the policy does not change from one iteration to the next, and
2. the length bound is not used to prune any search path during the last iteration.

This modified convergence test makes it possible to prove convergence to an optimal policy after a finite number of iterations, under the assumption that it is optimal to acquire perfect information at finite intervals.

Theorem 2 *If it is optimal to take an action that provides perfect information at finite intervals, then policy iteration using heuristic multi-step backups and the modified convergence test converges after a finite number of iterations to an optimal policy.*

Proof. There is no limit to how far the length bound can be increased over successive iterations. Because the optimal action sequence for every state is finite, there is some length bound such that an optimal policy and value function for action sequences up to that length bound is also optimal for action sequences of unbounded length. All we must show is that given a value function that is optimal for action sequences of unbounded length, there is some finite length bound such that all search paths are pruned by the value of perfect information heuristic.

The proof is by contradiction. Assume that for some state s, some path through the search tree of action sequences is never pruned. That is, for every subsequence $a_1..a_k$ of some infinite sequence of actions that do not provide information, $f(sa_1..a_k) > V^*(s)$, and so for $k = 1, 2, 3, \ldots$,

$$r(s, a_1) + \sum_{i=2}^{k} \beta^i p(s, a_1..a_{i-1}, s') r(s', a_i) + \beta^k \sum_{s'} p(s, a_1..a_k, s') V^*(s') > V^*(s).$$

But this implies that

$$r(s, a_1) + \sum_{i=2}^{\infty} \beta^i p(s, a_1..a_{i-1}, s') r(s', a_i) > V^*(s),$$

which contradicts the assumption that it is optimal to acquire perfect information at finite intervals. □

3.5 Adding actions that provide imperfect information

The results presented so far can be extended to POMDPs that include actions that provide imperfect information with the following restriction: the action set must still include some action (or actions) that provides perfect information, and an action that provides perfect information must still be taken at finite intervals.

Instead of searching an OR tree to find the best action sequence for each state in the policy improvement step, the algorithm is modified to search an AND/OR tree to find the best conditional plan or "solution tree." (See Nilsson, 1980, for a discussion of heuristic search of AND/OR trees.) In an AND/OR tree, an OR node corresonds to a choice of the best action to take and an AND node corresponds to a set of possible observations. All terminal nodes of a solution tree must still provide perfect information. Again, the search can be best-first or depth-first branch and bound. The heuristic evaluation function is the same, and the theorems already proved are easily generalized (although the change from action sequences to action trees makes the notation more cumbersome).

4 Example

To illustrate the performance of this algorithm, we describe a simple path-planning problem that is easily formalized as an MDP. This example is similar to "grid world" problems used by Artificial Intelligence researchers to study algorithms for planning and learning (e.g., Barto, Sutton, and Watkins, 1990).

Imagine a robot in the grid world shown in Figure 4 must find its way to a goal location, which is in the upper left-hand corner of the grid. Each cell of the grid corresponds to a state, with the states numbered for convenient reference. The robot has four actions it can take to move about the grid: it can move north (N), south (S), east (S), or west (W), one cell at a time. The robot can also take a stop action (X) once it reaches the goal state. (We assume the stop action moves the robot to an absorbing state that has zero cost.)

To complicate this path-planning problem, the robot's actions have unpredictable effects. If the robot attempts to move in a particular direction, it succeeds with probability 0.8. With probability 0.05 it moves in a direction that is 90 degrees off to one side of its intended direction, with probability 0.05 it moves in a direction that is 90 degrees off to the other side of its intended direction, and with probability 0.1 it does not move at all. If its movement would take it outside the grid, we consider it to have "bumped into a wall" and it remains in the same cell. The actions the robot takes to move about the grid provide no information, but it can choose an *observe* action (O) that provides perfect information about its current state. By interleaving the observe action with move actions, the robot can monitor its progress toward the goal. We assume the observe action incurs a cost of 1 and each move action also incurs a cost of 1, creating an incentive to reach the goal by as direct a route as possible. We also assume a cost of 5 is incurred if the robot bumps into a wall. If the robot stops in the goal state, it receives a reward of 100. Therefore the robot maximizes its return by finding a minimal-cost path to the goal state. We can use a discount factor of 1.0 because the robot eventually stops and enters the absorbing state. Table 1 records the

optimal action sequence for each state. Note that the interval between observations varies from state to state in an intuitively reasonable way.

Our value of perfect information heuristic makes it possible to solve this particular problem very efficiently. Approximately 2500 different information states are evaluated when the search heuristic is used to prune the search space, whereas exhaustive search up to a length limit of 12 – just long enough to find the same policy – would have to evaluate well over a billion information states per iteration. Policy iteration with heuristic multi-step backups converges to an optimal policy in eleven seconds on a DEC Alpha 3000/600.

To get some sense of how the efficiency of the algorithm might be affected by the size of the state set, we tested the algorithm on a series of similar gridworlds of increasing size and compared the time it takes conventional policy iteration to converge, assuming each move action provides perfect information, to the time it takes policy iteration with heuristic multi-step backups to converge to an optimal policy for interleaving move actions with costly observations. For these gridworlds, the length of an optimal action sequence in a state ranged from two to fifteen with an average of roughly seven. So there is a significant combinatorial explosion of possible action sequences. Our results are shown in Table 2. For this example at least, they show that the efficiency of heuristic multi-step backups is insensitive to the size of the state set. In fact, the additional time it takes to run policy improvement with heuristic multi-step backups grows slower as a function of the number of states than does the cost of policy evaluation – which dominates the cost of policy improvement as the number of states grows.

However, the efficiency of heuristic multi-step backups is very sensitive to other factors. This can be demonstrated by varying the cost of the observe action in our gridworld example. If the cost is zero, it is optimal to observe after every move action and the problem is equivalent to a completely observable MDP and easily solved. As the cost of the observe action is increased, the value of perfect information heuristic becomes less powerful and prunes less of the search space in multi-step backups. (The search heuristic becomes less powerful because it assumes perfect information is available at no cost, an assumption that becomes less accurate as the cost of the observe action increases.) Our results are shown in Table 3. Increasing the cost of the observe action moves the POMDP closer to the boundary of this subset of POMDPs for which it is optimal to acquire perfect information at finite intervals. When the cost of the observe action becomes sufficiently high, it is no longer optimal to observe after a finite interval (for at least one state) and the algorithm never converges. As increasing the observe cost moves the POMDP close to this boundary, the time it takes to converge begins to explode.

Although our algorithm converges to an optimal policy for any POMDP for which it is optimal to acquire perfect information at finite intervals, our empirical results suggest that not all problems in this subset of POMDPs can be solved with equal efficiency – at least not with this search heuristic. In particular, problems near the boundary of this subset of POMDPs may still be difficult to solve optimally. Nevertheless, our results do show that many problems in this subset of POMDPs can be solved efficiently with this algorithm, including POMDPs with hundreds of states. To put this in perspective, problems of this size are far beyond the range of current algorithms for general POMDPs which at present

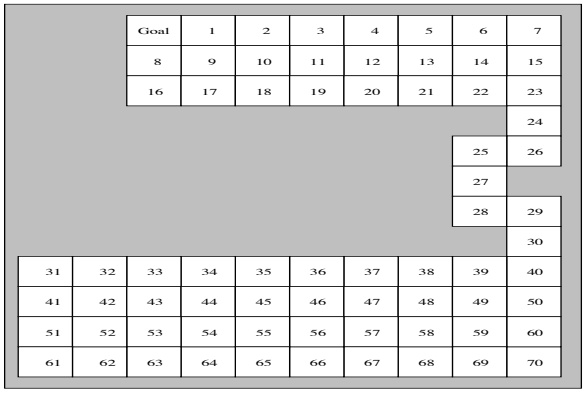

Figure 4: The states of the grid world are numbered for reference to Table 1.

State	Action Sequence	State	Action Sequence	State	Action Sequence	State	Action Sequence
Goal	X	18	NNWWO	36	EEEEO	54	ENEEEEEO
1	WO	19	NNWWWO	37	EEEO	55	ENEEEEO
2	WWO	20	NWWWWO	38	EEO	56	ENEEEO
3	WWWO	21	NWWWWWO	39	EO	57	ENEEO
4	WWWWO	22	NWWWWWWO	40	NNO	58	ENEO
5	WWWWWO	23	WNWWWWWWO	41	EEEEEEEEEO	59	NNEO
6	WWWWWWO	24	NNNWWWWWWWO	42	EEEEEEEEO	60	NNNNO
7	WWWWWWWO	25	EO	43	EEEEEEEO	61	ENENEEEEEEEO
8	NO	26	NNNO	44	EEEEEEO	62	NNEEEEEEEEO
9	WNO	27	NO	45	EEEEEO	63	NNEEEEEEEO
10	WWNO	28	NNO	46	EEEEO	64	NNEEEEEEO
11	WWWNO	29	WO	47	EEEO	65	NNEEEEEO
12	WWWWO	30	NO	48	EEO	66	NNEEEEO
13	WWWWWO	31	EEEEEEEEEO	49	NEO	67	NNEEEO
14	WWWWWWO	32	EEEEEEEEO	50	NNNO	68	NNEEO
15	WWWWWWWO	33	EEEEEEEO	51	EENEEEEEEEO	69	NNNEO
16	NNO	34	EEEEEEO	52	ENEEEEEEEO	70	NNNNNO
17	NNWO	35	EEEEEO	53	ENEEEEEEO		

Table 1: Optimal multi-step policy for the gridworld of Figure 4.

Number of states	50	100	150	200	250	300
Standard policy iteration	1	4	13	30	59	103
Multi-step policy iteration	11	28	65	136	252	428

Table 2: Time to convergence in seconds as problem size increases.

Cost for observing state	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.1	3.2	3.3	3.4	3.5
Time to converge in seconds	3	9	11	14	18	23	32	38	164	2569	10579	> 20000

Table 3: Algorithm efficiency as cost for observing state increases for gridworld in Figure 4.

cannot find optimal solutions to POMDPs with more than about fifteen or twenty states, and even then can sometimes take hours to converge (Littman, Cassandra, & Kaelbling, 1995). For the subset of POMDPs we have identified, multi-step policy iteration appears to offer the most efficient method available for finding an optimal policy.

5 Conclusion

The algorithm described in this paper combines dynamic programming and heuristic search. Heuristic search, in the form of heuristic multi-step backups, makes it possible to find an optimal policy without evaluating all information states. When it is optimal to take an action that provides perfect information at finite intervals, the number of information states visited by an optimal policy is finite. For such problems, we have shown that an optimal policy can be found by evaluating a finite number of information states even though an uncountably infinite number of information states are possible. A search heuristic is used to prune the rest of the state space.

Although the policy iteration algorithm we have described can only find optimal policies for POMDPs for which it is optimal to acquire perfect information at finite intervals, this limitation is offset by two advantages. For most problems in this class, it is faster than current algorithms for general POMDPs and makes it possible to solve larger problems. In addition, we have proved that it converges to an optimal policy after a finite number of iterations. At present, no other dynamic programming algorithm for POMDPs has been proved to converge after a finite number of iterations for POMDPs in this class.

Finally, the approach taken in this paper may provide insights that can help solve more general classes of POMDPs.

Acknowledgements

Thanks to Andy Barto, Michael Littman and Shlomo Zilberstein for helpful comments. This research was supported in part by the National Science Foundation under grants IRI-9624992 and ECS-9214866 and in part by Rome Laboratory, USAF, under grant F30602-95-1-0012.

References

[1] Abramson, B. (1993) A decision-theoretic framework for integrating sensors in AI plans. *IEEE Transactions on Systems, Man, and Cybernetics* 23:366-373.

[2] Barto, A.G.; Sutton, R.S.; and Watkins, C.J.C.H. (1990) Learning and Sequential Decision Making. In *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, M. Gabriel and J. Moore, (eds.), MIT Press, Cambridge, MA, 539-602.

[3] Hansen, E.A. (1994) Cost-Effective sensing during plan execution. In *Proceedings of the Twelth National Conference on Artificial Intelligence*, 1029-1035. Seattle, AAAI Press.

[4] Howard, R.A. (1960) *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA.

[5] Klein, M. (1962) Inspection-Maintenance-Replacement Schedules under Markovian Deterioration. *Management Science* 9:25-32.

[6] Langley, P.; Iba, W.; and Shrager, J. (1994) Reactive and Automatic Behavior in Plan Execution. In *Proceedings of the Second International Conference on Planning Systems*, 299-304. Chicago: AAAI Press.

[7] Littman, M.L.; Cassandra, A.R.; and Kaelbling, L.P. (1995) Efficient dynamic programming updates in partially observable Markov decision processes. Brown University Technical Report CS-95-19.

[8] Lovejoy, W.S. (1991) A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research* 28(1):47-65.

[9] Luss, H. (1976) Maintenance Policies When Deterioration Can Be Observed by Inspections. *Operations Research* 24:359-366.

[10] Monahan, G.E. (1982) A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science* 28(1):1-16.

[11] Nilsson, N. (1980) *Principles of Artificial Intelligence.* Palo Alto, CA: Morgan Kaufman.

[12] Puterman, M.L. and Shin, M.C. (1982) Action elimination procedures for modified policy iteration algorithms. *Operations Research* 30:301-318.

[13] Rosenfield, D. (1976) Markovian Deterioration with Uncertain Information. *Operations Research* 24:141-155.

[14] Ross, S.M. (1971) Quality Control under Markovian Deterioration. *Management Science* 17:587-596.

[15] Sheridan, T.B. (1970) On how often the supervisor should sample. *IEEE Transactions on Systems Science and Cybernetics* SSC-6:140-145.

[16] 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.

[17] Tijms, H.C. and Van Der Duyn Schouten, F.A. (1984) A Markov Decision Algorithm for Optimal Inspections and Revisions in a Maintenance System with Partial Information. *European Journal of Operational Research* 21:245-253.

[18] White, C.C. (1978) Optimal Inspection and Repair of a Production Process Subject to Deterioration. *Journal of the Operational Research Society* 29:235-243.