Approximate Solutions to Factored Markov Decision Processes via Greedy Search in the Space of Finite State Controllers

Kee-Eung Kim, Thomas L. Dean
Department of Computer Science
Brown University
Providence, RI 02912
{kek,tld}@cs.brown.edu

Nicolas Meuleau
MIT Artificial Intelligence Lab
545 Technology Square
Cambridge, MA 02139
nm@ai.mit.edu

Abstract

In stochastic planning problems formulated as factored Markov decision processes (MDPs), also called dynamic belief network MDPs (DBN-MDPs) (Boutilier, Dean, & Hanks 1999), finding the best policy (or conditional plan) is NP-hard. One of the difficulties comes from the fact that the number of conditionals required to specify the policy can grow to be exponential in the size of the representation for the MDP. Several recent algorithms have focused on finding an approximate policy by restricting the representation of conditionals using decision trees. We propose an alternative policy representation for Factored MDPs in terms of finite-state machine (FSM) controllers. Since practically speaking we are forced to limit the number of conditionals, we claim that there is a benefit to be had in using FSM controllers given that these controllers can use their internal state to maintain context information that might otherwise require a large conditional table or decision tree. Although the optimal policy might not be representable as a finite-state controller with a fixed amount of memory, we will be satisfied with finding a "good" policy; to that end, we derive a stochastic greedy-search algorithm based on recent developments in reinforcement learning (Baird & Moore 1999) and then demonstrate its performance in some example domains.

Introduction

Although remarkable advances in deterministic planning problems have been made and efficient algorithms are used in practice, stochastic planning problems for complex domains with large state and action spaces remain very hard to solve. One part of the difficulty is due to the fact that, although such problems can be compactly represented as factored Markov decision processes (MDPs), also called dynamic belief network MDPs (DBN-MDPs) (Boutilier, Dean, & Hanks 1999), the best policy might be impractical to represent. Most of the existing algorithms are based on Q-functions. Formal definitions will follow, but roughly speaking, the Q-functions determine the quality of executing actions in particular states. Several existing algorithms for factored MDPs rely approximating Q-functions to reduce the size of the representation — the hope is that these approximate Q-functions will ignore small differences among the values associated with action / state pairs thereby compressing the representation of the resulting policy. If these differences are indeed small the resulting policy should be close to optimal.

Most of the current algorithms assume a decision-tree representation for policies; intermediate nodes represent important aspects (or features) of the current state and leaf nodes determine which action to execute given those features are present in the current state (Boutilier, Dearden, & Goldszmidt 1995). The resulting class of policies are said to be *stationary* which is also in the class of *history independent* or *Markov* policies. Note, however, that by limiting the features used to condition actions and thereby restricting the size of the resulting policies, we also reduce our ability to discriminate among states. What this means is that the underlying dynamics (which is in part determined by the actions selected according to the agent's policy) becomes *non-Markovian*. Within the space of all policies that have the same capability with respect to discriminating among states, the best policy may require remembering features from previous states, i.e., the optimal policy may be history dependent. This observation is the primary motivation for the work described in this paper.

One way to represent the history is to use memory of past states. We could of course enhance the agent's ability to discriminate among states by having it remember features from previous states. However, this simply transforms the problem into an equivalent problem with an enlarged set of features exacerbating the curse of dimensionality in the process. An alternative approach that has been pursued in solving partially observable MDPs (POMDPs) is to use finite-state machine (FSM) controllers to represent policies (Hansen 1998; Meuleau *et al.* 1999). The advantage of this approach is that such controllers can "remember" features for an indefinite length of time without the overhead of remembering all of the intervening features. Search is carried out in the space of policies that have a finite

amount of memory.

It is important to note than in a (fully observable) MDP, the state contains all of the information required to choose the optimal action. Unfortunately, the current state may not encode this information in the most useful form and decoding the information may require considerable effort. In some domains, however, there is information available in previous states that is far more useful (in the sense of requiring less computational effort to decode for purposes of action selection) than that available in the present state. For example, in New York city it is often difficult to determine that gridlock has occurred, is likely to occur, or is likely to persist even if you have all of the information available to the Department of Transportation Traffic Control Center; however, if on entering the city around 3:00pm you note that traffic at the George Washington Bridge is backed up two miles at the toll booths, and you find yourself still in the city around 5:00pm, your best bet is to relax, do some shopping, have dinner in the city, and generally avoid spending several frustrating hours fuming in your car on the Triborough Bridge. We will return to a more formal example in a later section.

There are recent algorithms for reinforcement learning that assume the use of finite-state controllers for representing policies (Hansen 1998; Meuleau *et al.* 1999). These algorithms are intended to solve POMDPs and non-Markov decision problems. We argue that, as described in the previous paragraphs, the policy we search for should be history dependent if the problem is given as an MDP with a very large state space and the space of the policies determined by the agent's ability to discriminate among states does not cover the optimal memoryless policy.

In this paper, we adopt the approach of searching directly in the space of policies (Singh, Jaakkola, & Jordan 1994) as opposed to searching in the space of value functions, finding the optimal value function or a good approximation thereof and constructing a policy from this value function. Finding the best FSM policy is a hard problem and we side step some of the computational difficulties by using a stochastic greedy search algorithm (also adapted from work in reinforcement learning (Baird & Moore 1999)) to find a locally optimal finite state controller.

The inspiration for our approach came from the work of Meuleau et al. (1999) that restricted attention to the space of finite-state controllers and applied the VAPS family of algorithms (Baird & Moore 1999) to searching in the space of such policies. Our contribution is to show that the same basic ideas can be applied to solving factored MDPs with very large state spaces. There is nothing in principle preventing us from combining our work with that of Meuleau et al. to solve factored POMDPs in which a domain model is supplied. In this work, we see the availability of a factored domain model as an important advantage to be exploited.

The paper is organized as follows: We begin by formalizing stochastic planning problems as factored MDPs. We define the class of FSM policies that constitutes the space of possible solutions to the given problem. We then describe a local greedy search algorithm and investigate its performance by summarizing experimental results on examples of stochastic planning problems.

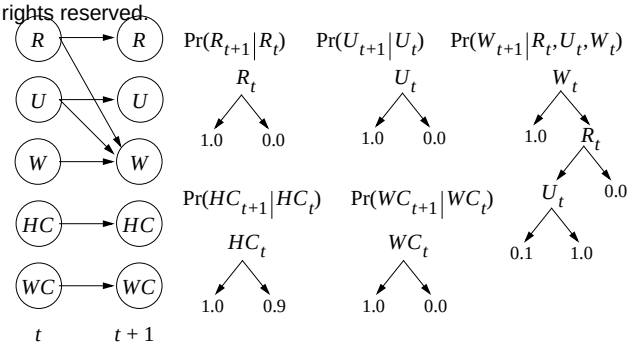

Figure 1: Compact representation of a robot action, GetCoffee.

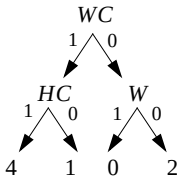

Figure 2: Compact representation of the reward function for the toy robot problem.

Factored MDPs

In this paper, we consider factored MDPs as input problems that generalize on propositional STRIPS problems. The factored MDP can be represented as a graphical model. An example is shown in Figure 1, which describes a toy robot planning problem. The ovals represent *fluents* in the domain. In this case, R, U, W, HC, WC are binary variables representing, respectively, the weather outside being rainy, the robot having an umbrella, the robot being wet, the robot holding coffee, and the robot's boss wanting coffee. The connections between the fluents at time step $t+1$ and t determine the dynamics. The conditional probability distribution governing state transitions is represented in terms of a set of conditional probability distributions, one for each fluent, shown as a tree in the right side of the Figure 1. The product of these distributions determines the state-to-state *transition probability function*. Since the dynamics of the state space depends on the action, we specify a graph representing the dynamics for each action. The figure shows the graph for one particular action, GetCoffee. To complete the specification, the reward function is also provided in a decision tree format. In Figure 2, we show an example of the reward function associated with the toy robot problem.

Definition 1 (Factored MDP) *A factored MDP $M = \{\vec{X}, A, T, R\}$ is defined as*

- $\vec{X} = [X_1, \ldots, X_n]$ *is the set of fluents that defines the state space. We use the lowercase letter* $\vec{x} = [x_1, \ldots, x_n]$ *to denote a particular instantiation of the fluents.*
- A *is the set of actions.*
- T *is the set conditional probability distributions, one for each action:*

$$T(\vec{x}_t, a, \vec{x}_{t+1}) = \prod_{i=1}^{n} P(x_{i,t+1} | \mathrm{pa}(x_{i,t+1}), a)$$

  where $\mathrm{pa}(X_{i,t+1})$ *denotes the set of parent variables of* $X_{i,t+1}$ *in the graphical model. Note that* $\forall i, \mathrm{pa}(X_{i,t+1}) \subseteq \{X_{1,t}, \ldots, X_{n,t}\}$.
- $R : \vec{X} \rightarrow \Re$ *is the reward function. Without loss of generality, we define the reward to be determined by both state and action* ($R : \vec{X} \times A \rightarrow \Re$).

□

We define *an explicit randomized history-independent policy* π as a function from $\vec{X} \times A$ to $[0, 1]$. In words, π defines a probability distribution over actions given that the system is in a specific state. Note that the size of the table for storing the probability distribution is exponential in the number of fluents (hence the term *explicit*). Every Factored MDP is associated with an objective function. In this paper, we restrict our attention to one particular objective function, *infinite horizon expected cumulative discounted reward* with discount rate $0 < \gamma < 1$. To compare the quality of policies, we define the following functions.

Definition 2 (Value Functions and Q-Functions) *The value function of the explicit history-independent policy π is defined as*

$$V^{\pi}(\vec{x}) = \sum_{a} \pi(\vec{x}, a) \big[R(\vec{x}, a) + \gamma \sum_{\vec{x'}} T(\vec{x}, a, \vec{x'}) V^{\pi}(\vec{x'}) \big].$$

The Q-function of the policy π is defined as

$$Q^{\pi}(\vec{x}, a) = R(\vec{x}, a) + \gamma \sum_{\vec{x'}} T(\vec{x}, a, \vec{x'}) V^{\pi}(\vec{x'}).$$

□

In words, $V^{\pi}(\vec{x})$ denotes the expectation of total discounted reward throughout the future starting from the state $\vec{x}$ and following the policy π. $Q^{\pi}(\vec{x}, a)$ denotes the expectation of total discounted reward throughout the future starting from the state $\vec{x}$, executing action a, and then following the policy π.

The optimal randomized explicit history-independent policy π^* is defined as

$$\pi^* = \arg\max_{\pi} V^{\pi}.$$

For the sake of simplicity, we denote the value function for the optimal policy π^* as V^* and call it the *optimal value function.*

We now ask ourselves whether there exists a *history-dependent policy* $\tilde{\pi}$ such that $V^{\tilde{\pi}} > V^*$. In the case of *explicit policies*, the answer is "no" due to the following theorem.

Theorem 1 (Page 143, Puterman (1994))

$$\max_{\tilde{\pi} \in \Pi^{\mathrm{HR}}} V^{\tilde{\pi}} = \max_{\pi \in \Pi^{\mathrm{MR}}} V^{\pi}$$

where Π^{HR} *and* Π^{MR} *represent the set of explicit randomized history-dependent policies and the set of explicit randomized history-independent policies, respectively.* □

A popular way to deal with the explosion in the size of the representation of π is to *aggregate* the atomic states that have same $V^{\pi}(\vec{x})$. Often, this is not enough, so we go further and aggregate the states that have *approximately same* value until the size of the representation becomes tractable. Unfortunately, this breaks the above theorem. It is easy to observe that by doing so, the problem becomes a POMDP. This justifies our claim that we should search in the space of history-dependent policies instead of history-independent policies. In the next section, we formally derive a special case in which a history-dependent policy results in an exponential saving in size compared to a history-independent policy.

Finite Policy Graphs (Conditional Plans)

A deterministic policy graph for a given Factored MDP is a directed graph $(\mathcal{V}, \mathcal{E})$ where each vertex $v \in \mathcal{V}$ is labeled with an action $a \in A$, and each arc $e \in \mathcal{E}$ is labeled with some boolean function $f_e(\vec{x})$. Also for each vertex, there is one and only one outgoing arc that satisfies the associated boolean function. When the controller is at a certain vertex, it executes the action associated with the vertex. This implies a state transition in the underlying MDP. Upon observing the new state $\vec{x}$, the controller takes the appropriate outgoing arc and arrives at the destination vertex.

In this paper, we focus on stochastic policy graphs in which the action choices and vertex transitions are probabilistic. Further, we assume that all the labeled boolean functions take only one of the n fluents as input.

Definition 3 (Unary Stochastic Policy Graphs) *A unary stochastic policy graph is defined by the following parameters.*

- $\mathcal{V}$ *is the set of vertices in the graph.*
- $\mathcal{E}$ *is the set of directed edges in the graph.*
- $\psi(v, a)$ *is the probability of choosing action a in vertex $v \in \mathcal{V}$:*

$$\psi(v, a) = P(A_t = a | \mathcal{V}_t = v)$$

  for all time steps t.
- $\eta((v, i), x_i, (v', i'))$ *is the joint probability of moving from vertex $v \in \mathcal{V}$ and focusing on the fluent X_i to*

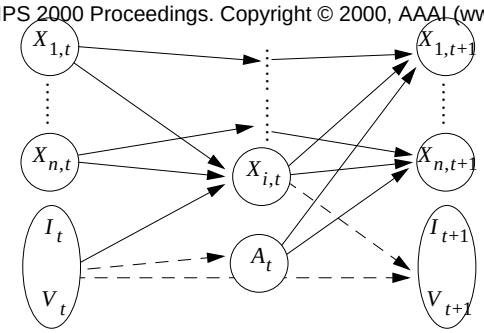

Figure 3: Influence diagram illustrating the policy graph coupled with a Factored MDP.

vertex $v' \in \mathcal{V}$ and focusing on the fluent $X_{i'}$, after observing $x_i \in X_i$:

$$\eta((v,i),x_i,(v',i')) = P(\mathcal{V}_{t+1}=v',\mathcal{I}_{t+1}=i'|\mathcal{V}_t=v,\mathcal{I}_t=i,X_{i,t+1}=x_i)$$

for all time steps t.

- *$\eta_{\mathcal{I}_0}(i)$ is the probability distribution for focusing on the fluent X_i at time step 0.*

$$\eta_{\mathcal{I}_0}(i) = P(\mathcal{I}_0 = i)$$

- *$\eta_{\mathcal{V}_0}(i,x_i,v)$ is the probability distribution for the initial vertex v_0 conditioned on the event that the initial focus is on the fluent X_i and its value is x_i:*

$$\eta_{\mathcal{V}_0}(i,x_i,v) = P(\mathcal{V}_0 = v|\mathcal{I}_0 = i, X_{i,0} = x_i)$$

□

Figure 3 shows the diagram of the policy graph. All the dotted links represent the parameters of the policy. There is a range of possible structures for the policy. We can make the policy *reactive* in the sense that there is no memory associated with the finite-state controller, i_t is fixed constant all the time, and the distribution on a_t is determined by $x_{i,t}$ only. HQL-type policies (Wiering & Schmidhuber 1997), that consist of a sequence of reactive policies, can be modeled as well, by having $|\mathcal{V}|$ as large as the number of reactive policies and associating goal state and fluent index with the vertex in the policy graph.

Now we might ask what kind of leverage we can get by using this kind of history-dependent policies? Consider the problem given as a Factored MDP shown in Figure 4 and Figure 5. The Factored MDP is constructed in such a way that its state is a randomly generated instance of the majority problem and the reward is given when the correct answer is given as the action. Thus, the optimal policy is to generate the answer corresponding to whether the majority of bits in a random n-bit vector is set or not. The Factored MDP representation is composed of $n+2$ variables, where $X_1,\ldots,X_n$ represent the n-bit vector encoding the problem instance, Y represents the *answer* to the problem (yes or no),

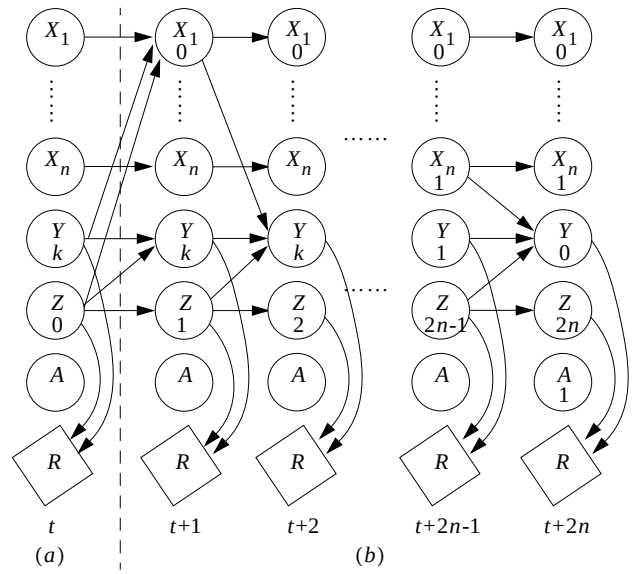

Figure 4: A task determining majority represented as a Factored MDP. (a) answer generation phase : In this example, Y is sampled to be k at time t. (b) problem instance generation phase : Then, exactly k fluents among $X_1,\ldots,X_n$ are set to 1 by the time $t+2n$. In this example, X_1 is sampled to be 1 and X_n to be 0.

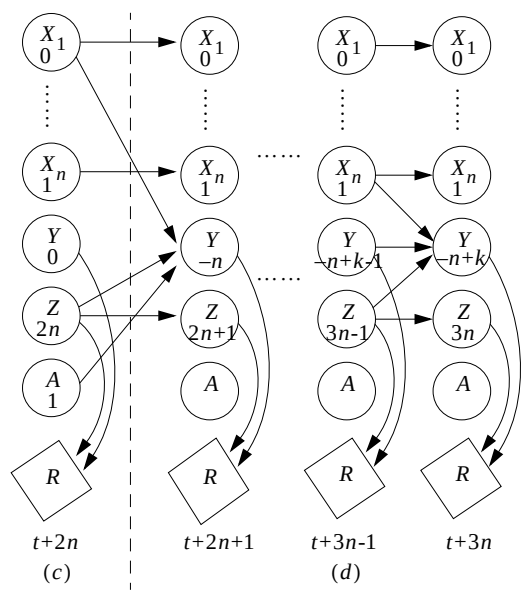

Figure 5: A task determining majority represented as a Factored MDP (continued from Figure 4). (c) answer guess phase: In this example, at time $t+2n$, the guess "yes" is fed into the Factored MDP. (d) answer verification phase: The guess answer is checked and the reward of 1.0 will be given at time $t+3n$ if and only if $-n/2 < -n+k \leq n/2$.

and Z represents a counter for keeping track of the four *phases* of the process:

$$Z = \begin{cases} 0 & \text{answer generation phase,} \\ z \in [1, 2n] & \text{problem instance generation phase,} \\ 2n & \text{answer guess phase,} \\ z \in [2n+1, 3n] & \text{answer verification phase.} \end{cases}$$

The dynamics of the Factored MDP is described as follows. Unless otherwise specified, all fluents except Z retain values from the previous time step, and Z is increased at each time step:

Answer generation phase : Randomly generate an answer.

[**Time** t] $X_{1,t}, \ldots, X_{n,t}$ retain values from the previous time step. Y_t is set to a random number $y_t \in [1, n]$. Z_t is set to 0.

Problem instance generation phase : Randomly generate a problem instance corresponding to the answer generated in the previous phase.

[**Time** $t+1 \sim t+2n$] For $1 \leq i \leq n$, $X_{i,t+2i-1}$ is set to one with probability $\min[y_{i,t+2i-2}/(n-i+1), 1]$ at time $t+2i-1$. Y_{t+2i} is set to $y_{t+2i-1} - x_{i,t+2i-1}$. In short, throughout the problem generation phase, the MDP generates a random n-bit vector which has y_t bits set to 1, and Y serves as a counter keeping track of how many more bits should be set.

Answer guess phase : The answer is guessed by the current policy.

[**Time** $t+2n$] An action is taken, which corresponds to the guess whether the majority of the $X_1, \ldots, X_n$ are set or not. 0 means the answer "no" and 1 means "yes".

Answer verification phase : The answer is verified by counting the number of fluents that are set to 1 among fluents $X_1, \ldots, X_n$.

[**Time** $t+2n+1$] Y_{t+2n+1} is set to $a_{t+2n} * (-n) + x_{1,t+2n}$.

[**Time** $t+2n+2 \sim t+3n$] For $2 \leq i \leq n$, Y_{t+2n+i} is set to $y_{t+2n+i-1} + x_{i,t+2n+i-1}$.

[**Time** $t+3n$] If $-n/2 < Y_{t+3n} \leq n/2$, the guess was correct, so the reward of 1 is given. Otherwise, no reward is given.

[**Time** $t+3n+1$] The system gets back to the answer generation phase (time t).

A history-independent policy represented as a tree should be able to count the number of bits set among $X_1, \ldots, X_n$. It will, however, explode the size of the tree. On the other hand, a history-dependent policy with sufficient amount of memory (size of 2 in the above case) can remember the answer. Hence, the latter will not explode.

Local Greedy Policy Search

Searching in the space of history-dependent policies does not make the problem any easier. In this paper, we use a local greedy search method based on Baird and Moore's VAPS algorithm (Baird & Moore 1999) and its predecessor REINFORCE algorithm (Williams 1988). It is a trial-based, stochastic gradient descent of a general error measure, and hence we can show that it converges to a local optimum with probability 1. The error measure we use in this paper is the expected cumulative discounted reward:

$$B_\pi = \mathbf{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(\vec{x}_t, a_t) | \vec{x}_0, \pi\Big]$$

Assume that the problem is a goal-achievement task — there exists an absorbing goal-state which the system must be in as fast as possible. As soon as the system reaches the goal-state, the system halts and assigns a reward. In this case, we can write down our optimality criterion as

$$C_\pi = \sum_{T=0}^{\infty} \sum_{h_T \in H_T} P(h_T | \vec{x}_0, \pi) \epsilon(h_T), \tag{1}$$

where H_T is the set of all trajectories that terminate at time T, *i.e.*,

$$h_T = [\vec{x}_0, v_0, i_0, a_0, r_0, \ldots, \vec{x}_T, v_T, i_T, a_T, r_T]$$

and $\epsilon(h_T)$ denotes the total error associated with trajectory h_T. We assume the total error ϵ is additive in the sense that

$$\epsilon(h_T) = \sum_{t=0}^{T} e(h_T[0, t])$$

where $e(h_T[0, t])$ is an instantaneous error associated with sequence prefix

$$h_T[0, t] = [\vec{x}_0, v_0, i_0, a_0, r_0, \ldots, \vec{x}_t, v_t, i_t, a_t, r_t]$$

Among the many ways of defining e, we use TD(1) error (Sutton & Barto 1998) which is

$$e(h_T[0, t]) = -\gamma^t r_t.$$

In this case, we note that our objective function (Equation 1) is exactly $-B_\pi$, therefore we try to minimize C_π.

To illustrate the stochastic gradient descent algorithm on C_π, we derive the gradient ∇C_π with respect to the parameters of the policy graph. From Definition 3, the parameter space is $\{\psi, \eta, \eta_{\mathcal{I}_0}, \eta_{\mathcal{V}_0}\}$. Without loss of generality, let ω be any parameter of the policy graph. Then we have

$$\frac{\partial C_\pi}{\partial \omega} = \sum_{T=0}^{\infty} \sum_{h_T \in H_T} \Big[P(h_T | \vec{x}_0, \pi) \frac{\partial \epsilon(h_T)}{\partial \omega} + \epsilon(h_T) \frac{\partial P(h_T | \vec{x}_0, \pi)}{\partial \omega} \Big]. \tag{2}$$

Note that for our choice of ϵ the partial derivative of $\epsilon(h_T)$ w.r.t. ω is always 0. We now derive the partial derivative of $P(h_T|\vec{x}_0,\pi)$ w.r.t. ω. Since

$$\begin{aligned} P(h_T|\vec{x}_0,\pi) =& P(\vec{x}_0)\eta_{\mathcal{I}_0}(i_0)\eta_{\mathcal{V}_0}(i_0,x_{i,0},v_0)\psi(v_0,a_0) \\ & \prod_{t=1}^{T}\Big[T(\vec{x}_{t-1},a_{t-1},\vec{x}_t) \\ & \quad \eta((v_{t-1},i_{t-1}),x_{i,t},(v_t,i_t))\psi(v_t,a_t)\Big], \end{aligned}$$

we can rewrite Equation 2 as

$$\begin{aligned} \frac{\partial C_\pi}{\partial \omega} = \sum_{T=0}^{\infty}\sum_{h_T\in H_T} P(h_T|\vec{x}_0,\pi)\Big[&\epsilon(h_T)\sum_{t=0}^{T}\frac{\partial \ln\psi(v_t,a_t)}{\partial\omega} \\ &+\epsilon(h_T)\sum_{t=1}^{T}\frac{\partial \ln\eta((v_{t-1},i_{t-1}),x_{i,t},(v_t,i_t))}{\partial\omega} \\ &+\epsilon(h_T)\frac{\partial \ln\eta_{\mathcal{I}_0}(i_0)}{\partial\omega} \\ &+\epsilon(h_T)\frac{\partial \eta_{\mathcal{V}_0}(i_0,x_{i,0},v_0)}{\partial\omega}\Big]. \end{aligned}$$

The above equation suggests that the gradient is the mean of

$$\begin{aligned} g(h_T) = &\epsilon(h_T)\sum_{t=0}^{T}\frac{\partial \ln\psi(v_t,a_t)}{\partial\omega} \\ &+\epsilon(h_T)\sum_{t=1}^{T}\frac{\partial \ln\eta((v_{t-1},i_{t-1}),x_{i,t},(v_t,i_t))}{\partial\omega} \\ &+\frac{\partial \ln\eta_{\mathcal{I}_0}(i_0)}{\partial\omega} \\ &+\frac{\partial \eta_{\mathcal{V}_0}(i_0,x_{i,0},v_0)}{\partial\omega} \end{aligned} \tag{3}$$

w.r.t. the distribution $P(h_T|\vec{x}_0,\pi)$. Hence, stochastic gradient descent of the error is done by repeatedly sampling a trajectory h_T and evaluating $g(h_T)$ and updating the parameters of the policy graph.

It is important that we have non-zero probabilities for all possible trajectories. One way to enforce this condition is to use a *Boltzmann distribution* for the parameters of the policy graph:

$$\begin{aligned} \psi(v,a) &= \frac{e^{Q_\psi(v,a)}}{\sum_{a'\in A} e^{Q_\psi(v,a')}} \\ \eta((v,i),x_i,(v',i')) &= \frac{e^{Q_\eta((v,i),x_i,(v',i'))}}{\sum_{v''\in\mathcal{V},i''} e^{Q_\eta((v,i),x_i,(v'',i''))}} \\ \eta_{\mathcal{I}_0}(i) &= \frac{e^{Q_{\eta_{\mathcal{I}_0}}(i)}}{\sum_{i'} e^{Q_{\eta_{\mathcal{I}_0}}(i')}} \\ \eta_{\mathcal{V}_0}(i,x_i,v) &= \frac{e^{Q_{\eta_{\mathcal{V}_0}}(i,x_i,v)}}{\sum_{v'\in\mathcal{V}} e^{Q_{\eta_{\mathcal{V}_0}}(i,x_i,v')}} \end{aligned}$$

Throughout the experiments described in the next section, we use the above reparameterization.

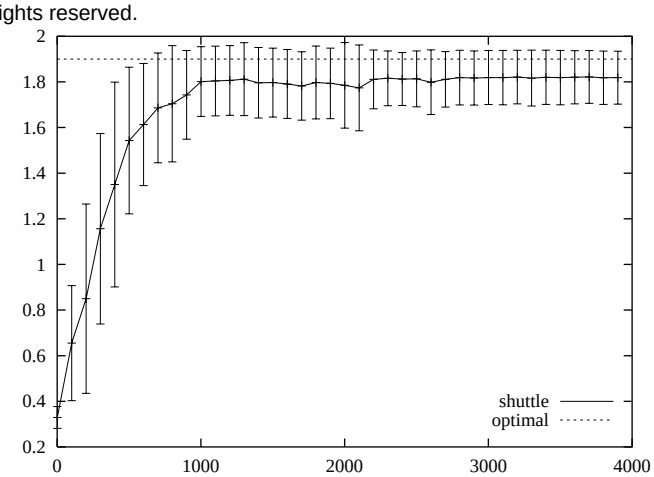

Figure 6: Performance on the modified shuttle problem.

Preliminary Experiments

In this section, we show some results from our preliminary experiments.

Figure 6 presents the learning curve of the policy graph on a modified version of a benchmark problem for evaluating POMDP algorithms called the "shuttle problem." The original shuttle problem (Cassandra 1998) is given as a POMDP, where the task is driving a shuttle back and forth between a loading site and an unloading site to deliver a package. The status of whether the shuttle is loaded or not is hidden. For our experiments, we modified the problem so that everything is observable — it is now composed of three fluents, that represent the location of the shuttle, the status of loaded or not, and the status of the package's arrival at the loading site, respectively. A reward of 1 is given every time the shuttle arrives at the loading site with empty cargo and returns to the unloading site with the package. With the discount rate of 0.9, the optimal performance is around at 1.9. The graph shows the average of 10 independent experiments each with 4000 gradient updates on a policy graph with 2 vertices.

Figure 7 shows the performance of the algorithm on the toy coffee robot problem. It is an extended version of the simple problem in Figure 1. When explicitly enumerated (*i.e.* flattened out), the problem has 400 states. By using a policy graph with 2 vertices, we were able to find a locally optimal policy within minutes. The optimal performance is 37.49. The graph shows the average of 10 independent experiments each with 4000 gradient updates. On average, the learned policy graphs had performance around at 25. We expected to learn a gradually better policy as we added more vertices, but we ended up stuck at local optima with same performance. The result is to be expected given that the algorithm looks for local optima; however, the result is in contrast to what we observed in

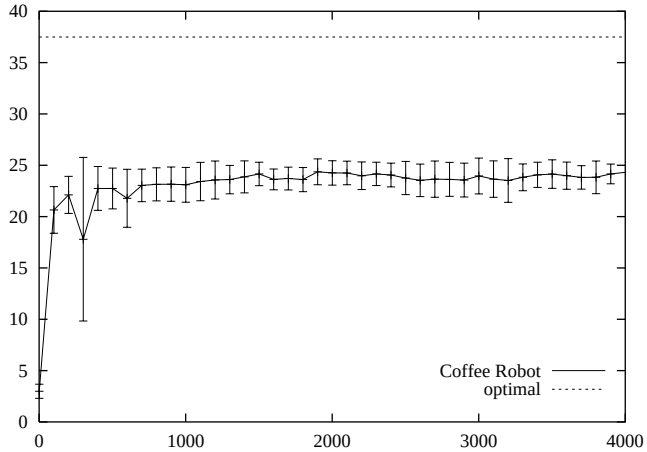

Figure 7: Performance on the Coffee Robot domain.

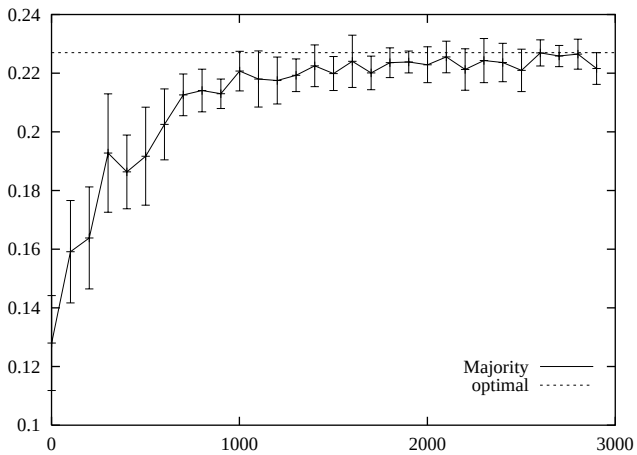

Figure 8: Performance on the Majority domain with 5 bits.

another paper (Meuleau *et al.* 1999) where increasing the size of the policy graphs allowed for gradual improvements. We suspect that there are large discrete jumps in the performance as the expressiveness of the policy passes some threshold. The analyses on such intriguing behavour remain as future work.

Figure 8 shows the performance on the majority domain shown in Figure 4 and Figure 5. The task was to determine the whether the majority of 5 bits are set or not. When explicitly enumerated, the domain has 5632 states. Using 1-bit memory (2 state FSM), the stochastic gradient descent algorithm converged to the optimum. If we were to use a history-independent policy represented as a decision tree, the most compact optimal policy will have 2^5 leaves. Again, the performance graph shows the average of 10 independent experiments each with 3000 gradient updates.

Related Work

There is a large and growing literature on the efficient representation and solution of problems involving planning under uncertainty modeled as Markov decision processes (see (Boutilier, Dean, & Hanks 1999) for a survey) that draws upon earlier work in operations research (see (Bertsekas 1987; Puterman 1994) for an overview). In particular, there has been much work recently representing large MDPs using compact, factored representations such as Bayesian networks (Boutilier, Dearden, & Goldszmidt 1995; Dean, Givan, & Leach 1997; Dean, Givan, & Kim 1998; Hauskrecht *et al.* 1998).

Roughly speaking, the computational complexity of solving MDPs is polynomial in the size of the state and action spaces (Littman, Dean, & Kaelbling 1995) and the typical factored MDP has state and action spaces exponential in the size of the compact representation of the dynamics (the state-transition and reward functions). Methods from reinforcement learning (see (Kaelbling, Littman, & Moore 1996; Sutton & Barto 1998) for surveys) and function approximation more generally (Bertsekas & Tsitsiklis 1996) have been studied as a means of avoiding the enumeration of states and actions. These learning methods do not require the use of an explicit model, instead requiring only a generative model or the ability to perform experiments in the target environment (Kearns, Mansour, & Ng 1999). Still, when available, explicit models can provide valuable information to assist in searching for optimal or near optimal solutions.

The work described in this paper borrows from two main threads of research besides the work on factored representations. The first thread follows from the idea that it is possible, even desirable in some cases, to search in the space of policies instead of in the space of value functions which is the basis for much of the work involving dynamic programming. The second thread is that it is possible to use gradient methods to search in the space of policies or the space of value functions if the parameterized form of these spaces is smooth and differentiable.

Singh et al. (1994) investigate methods for learning without state estimation by searching in policy space instead of searching for an optimal value function and then constructing an optimal (but not necessarily unique) policy from the optimal (but unique) value function or a reasonable approximation thereof. Baird and Moore (1999) introduce the VAPS (Value And Policy Search) family of algorithms that allow searching in the space of policies or in the space of value functions or in some combination of the two. Baird and Moore's work generalizes on and extends the work of Williams (1992).

Hansen (1998) describes algorithms for solving POMDPs by searching in the policy space of finite memory controllers. Meuleau et al. (1999) combines the idea of searching in an appropriately parameterized

policy space with the idea of using gradient based methods for reinforcement learning. Our work employs an extension of the VAPS methodology to factored MDPs.

Conclusion

We have described a family of algorithms for solving factored MDPs that works by searching in the space of history-dependent policies, specifically finite-state controllers, instead of the space of Markov policies. These algorithms are designed to exploit problems in which useful clues for decision making anticipate the actual need for making decisions and these clues are relatively easy to extract from the history of states and encode in the local state of a controller. Since MDPs assume full observability, all information necessary for selecting the optimal action is available encoded in the state at the time it is needed; in some cases, however, this information can be computationally difficult to decode and hence the anticipatory clues can afford a significant computational advantage.

This work builds on the work of Meuleau et al. (1999) in solving POMDPs and further extends the work of Baird and Moore (1999) to factored MDPs. We hope to show that our approach and that of Meuleau et al. can be combined to solve factored POMDPs for cases in which a domain model is provided and provides some insight into the underlying structure of the problem.

Acknowledgments This work was supported in part by the Air Force and the Defensse Advanced Research Projects Agency of the Department of Defense under grant No. F30602-95-1-0020. We thank Leslie Pack Kaelbling and Phil Klein for invaluable comments and Craig Boutilier for sharing the Coffee Robot domain.

References

Baird, L., and Moore, A. 1999. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*. Cambridge, MA: MIT Press.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Belmont, Massachusetts: Athena Scientific.

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

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proceedings IJCAI 14*, 1104–1111. IJCAII.

Cassandra, A. R. 1998. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. Ph.D. Dissertation, Department of Computer Science, Brown University.

Dean, T.; Givan, R.; and Kim, K.-E. 1998. Solving planning problems with large state and action spaces. In *Fourth International Conference on Artificial Intelligence Planning Systems*.

Dean, T.; Givan, R.; and Leach, S. 1997. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In Geiger, D., and Shenoy, P. P., eds., *Thirteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann.

Hansen, E. A. 1998. Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 211–219.

Hauskrecht, M.; Meuleau, N.; Boutilier, C.; Kaelbling, L. P.; and Dean, T. 1998. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 220–229.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285.

Kearns, M.; Mansour, Y.; and Ng, A. 1999. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *Proceedings IJCAI 16*. IJCAII.

Littman, M.; Dean, T.; and Kaelbling, L. 1995. On the complexity of solving Markov decision problems. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, 394–402.

Meuleau, N.; Peshkin, L.; Kim, K.-E.; and Kaelbling, L. 1999. Learning finite-state controllers for partially observable environments. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*.

Puterman, M. L. 1994. *Markov Decision Processes*. New York: John Wiley & Sons.

Singh, S.; Jaakkola, T.; and Jordan, M. 1994. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the Eleventh Machine Learning Conference*.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: MIT Press.

Wiering, M., and Schmidhuber, J. 1997. HQ learning. *Adaptive Behavior* 6(2):219–246.

Williams, R. J. 1988. Towards a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, Northeastern University, Boston, Massachusetts.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8:229–256.