A Survey of POMDP Solution Techniques

Kevin P. Murphy

9 September 2000

1 Introduction

One of the goals of AI is to design an agent[1] which can interact with an environment so as to maximize some reward function. Control theory addresses the same problem, but uses slightly different language: agent = controller, environment = plant, maximizing reward = minimizing cost. Control theory is mainly concerned with tasks in continuous spaces, such as designing a guided missile to intercept an airplane in minimum expected time, whereas AI is mainly concerned with tasks in discrete spaces, such as designing a program to play bridge to maximize the chance of winning. Nevertheless, AI and control theory have much in common [DW91], and some problems, such as designing a mobile robot to perform household chores, will require techniques from both fields.

When designing agents that can act under uncertainty, it is convenient to model the environment as a POMDP (Partially Observable Markov Decision Process, pronounced "pom-dp"). At (discrete) time step t, the environment is assumed to be in some state X_t. The agent then performs an action (control) A_t, whereupon[2] the environment (stochastically) changes to a new state X_{t+1}. The agent doesn't see the environment state, but instead receives an observation Y_t, which is some (stochastic) function of X_t. (If $Y_t = X_t$, the POMDP reduces to a fully observed MDP.) In addition, the agent receives a special observation signal called the reward, R_t.[3] The POMDP is characterized by the state transition function $P(X_{t+1}|X_t, A_t)$, the observation function $P(Y_t|X_t, A_{t-1})$, and the reward function $E(R_t|X_t, A_{t-1})$.

The goal of the agent is to learn a policy π which maps the observation history (trajectory)

$$h_{1:t} \stackrel{\text{def}}{=} ((Y_1, A_1, R_1), \ldots, (Y_{t-1}, A_{t-1}, R_{t-1}), (Y_t, -, -))$$

into an action A_t to maximize π's quality or value (see Figure 1). (In the case of a stochastic policy, $\pi(h_{1:t})$ returns a probability distribution over actions.) The value of a policy can be defined in several ways. One way is the expected discounted infinite sum of rewards, $V_\gamma(\pi) = E \sum_{t=1}^{\infty} \gamma^{t-1} R_t$, where $0 \leq \gamma < 1$ is a discount factor which ensures this sum is finite. (The expectation is taken over starting states and the stochastic transitions/emissions of the environment, and any stochastic decisions made by the agent.) A special case is where the agent is trying to get to a specific absorbing goal state (as in classical AI planning). In this case, we can give a reward of -1 per step except in the goal state, and set $\gamma = 1$: the cost of the policy becomes the expected time to reach the goal. For non-periodic tasks, the expected reward per time step is often a better measure: $V_{\text{avg}}(\pi) = E \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} R_t$.

Since the agent has finite memory capacity, it must somehow compress the unbounded history $h_{1:t}$ into an internal state, S_t, as illustrated in Figure 2. So now the agent must learn a policy that maps from (internal agent) states to actions. In addition, it must have a way of updating its internal state when a new observation arrives.

[1] We do not share Andrew Moore's view that the only proper use of the word "agent" is when precedeed by the words "travel", "secret" or "double" :)

[2] In this paper, we assume all actions take one unit of discrete time at some (unspecified) time scale. If we allow actions to take variable lengths of time, we end up with a semi-Markov model; see e.g., [SPS99].

[3] It might seem more natural to make the distinction between Y_t and R_t inside the agent. For instance, if the agent has different goals, the reward signal might depend on the agent's state as well as the environment's. The problem with this approach is that the agent might falsely reward itself e.g., if the perceived location matches the desired location, the agent rewards itself, even if it is making an error in its perception, or even if the desired location actually has no real utility for the agent (because it mistakenly believed there was food there, for instance). The agent *can* use internal ("shaping") rewards in its learning algorithm, but unless these satisfy certain constraints [NHR99] with respect to the true, external, objective reward function, this can lead the agent astray. For MDPs, there is essentially no agent/environment distinction, so it is possible to encode the agent's goal as an extra state variable, in which case the reward implicitely depends on it.

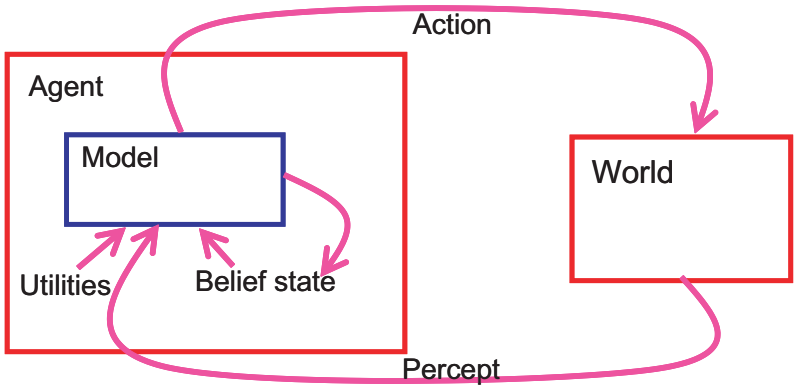

Figure 1: The percept-action cycle for a model-based agent. Note that the utility function of the agent (which defines the current task) is part of its internal state, not a property of the world.

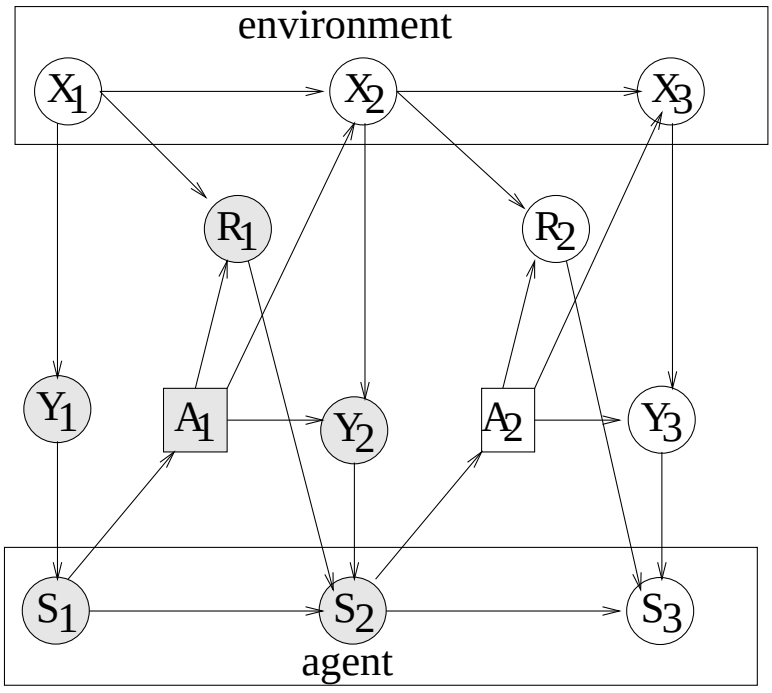

Figure 2: An agent interacting with an environment, modelled as two coupled finite state automata. The agent has seen history $h_{1:2} = (Y_{1:2}, A_1, R_1)$ and is about to decide on action A_2.

2 Types of agents

We can categorize agents (controllers) along the following axes; we will go into details later.

- Does the agent know the environment model or not? If not, does it learn one?
- What kind of internal state does the agent maintain?
  - $S_t = Y_t$, which defines a reactive (memoryless) agent.
  - $S_t = h_{t-k:t}$, a finite fixed-length window of past observations.
  - S_t is a suffix tree, which defines a variable-length Markov model.
  - S_t is a recurrent neural network.
  - $S_t = P(X_t|h_{1:t})$ is a belief state. (This requires knowing the environment model.)
  - S_t is a finite state machine.
- How is the policy represented and learned?
  - Learn $Q(s, a)$ using some reinforcement learning technique [SB98]. Note that this may not work since the environment might not be Markov in S. If s is continuous valued, we may need to use function approximators to represent Q.
  - If the POMDP is known, we can convert it to a belief-state MDP (see Section 3), and compute V for that. This is the optimal approach if the model is known, but is often computationally intractable. We can then consider either approximating V or the belief state, or both.
  - If the POMDP is known, we can solve the underlying (completely observed) MDP, and use that as the basis of various heuristics: see Section 3.5.
  - If we have the ability to compute $V(\pi)$ or $\frac{\partial V(\pi)}{\partial \theta_\pi}$, where θ_π are the parameters of the policy, then we can perform a (local) search for the best controller: see Section 5.

3 Belief state controllers

If the environment model is known, the optimal approach is for the agent to compute the sufficient statistic $b_t = P(X_t|h_{1:t})$, and use this as its internal state [KLC98]; b_t is called the belief (information) state, and can be updated online using Bayes rule (sometimes called a state estimator):

$$\begin{aligned} b_t(x) &= SE(b_{t-1}, a, y) = P(x|y, a, y_{1:t-1}, a_{1:t-1}) \\ &\propto P(y|x, a) \sum_{x'} P(x|a, x') b_{t-1}(x') \end{aligned}$$

Any given discrete POMDP induces a MDP whose states are belief states. The transition function of this MDP is

$$P(b'|a, b) = \sum_y P(b'|a, b, y) P(y|a, b)$$

where $P(b'|b, a, y) = \delta(SE(b, a, y), b')$ and

$$P(y|a, b) = \sum_x \sum_{x'} b(x') P(x|x', a) P(y|a, x)$$

is just (the reciprocal of) the normalizing factor of the SE equation. The reward function is

$$R(b, a) = \sum_x b(x) R(x, a)$$

If we can compute the value function for this belief-state MDP, $V(b)$, we can define our controller as $\pi(b) = \arg\max_a Q(b,a)$, where Q satisfies

$$\begin{aligned} Q(b,a) &= R(b,a) + \gamma \sum_{b'} P(b'|b,a) V(b') \\ &= \sum_x b(x) R(x,a) + \gamma \sum_y P(y|a,b) V(SE(b,a,y)). \end{aligned}$$

Substituting in the previous definitions we get

$$\begin{aligned} Q(b_t,a_t) &= E_{x_t}\left[R(x_t,a_t) + \gamma E_{x_{t+1}} E_{y_{t+1}} V(b_{t+1})\right] \\ &= \sum_{x_t} P(x_t|h_{1:t}) \left[R(x_t,a_t) + \gamma \sum_{x_{t+1}} P(x_{t+1}|x_t,a_t) \sum_{y_{t+1}} P(y_{t+1}|x_{t+1},a_t) V(b_{t+1}) \right] \end{aligned}$$

Given the belief state MDP, we can consider the following four options:

- Compute $V(b)$, i.e., the exact value function of the exact belief state.
- Compute $\hat{V}(b)$, i.e., an approximate value function of the exact belief state.
- Compute $V(\hat{b})$, i.e., the exact value function for an approximate belief state.
- Compute $\hat{V}(\hat{b})$, i.e., an approximate value function for an approximate belief state.

These approximations are necessary because computing both V and b can be intractable (computing V is doubly exponential in the horizon time, and computing b is exponential in the number of discrete state variables). We discuss all four combinations below.

3.1 Exact V, exact b

Although the states of the belief-state MDP are continuous valued, the value function turns out to be piecewise linear and convex (PLC). This can be used as the basis of exact algorithms, such as the Witness algorithm, for computing $V(b)$: see [KLC98, Cas98] for a review. (For a simple explanation of these algorithms, see `http://www.cs.brown.edu/research/ ai/pomdp/tutorial`.) Unfortunately, exact computation of $V(b)$ is intractable; current computing power can only solve POMDPs with a few dozen states.

[BP96] discuss a way to exploit structure in a POMDP (in the form of DBN) to compute exact tree-structured value functions and policies.

3.2 Approximate V, exact b

There have been many attempts to compute approximate value functions for the belief-state MDP, many of which just treat it like a regular continuous-state MDP. See [Hau00] for a review. We mention just a few here.

- [PR95] proposed a smooth, differentiable approximation to the PLC value function called SPOVA, defined as $V(b) = \left(\sum_{i=1}^N (\sum_x b(x)\phi_i(x))^k\right)^{1/k}$, where each ϕ_i is a vector of size $|b|$. The free parameters are N, the number of vectors, the ϕ_i vectors themselves, and k, the degree of smoothing.
- Discretizing b into a grid and using interpolation: see [Bra97].

3.3 Exact V, approximate b

Computing b can be inefficient. Two possible approximations that can be used are the Boyen-Koller (BK) algorithm [BK98] (which requires that the model be specified as a DBN), and particle filtering (PF) [DdFG01]. For a POMDP, the goal of state estimation is merely to be on the right side of the decision boundary of the policy. [PB00] suggest a way to modify sampling techniques to take this into account, to speedup runtime performance.

3.4 Approximate V, approximate b

[RPK99] use the BK algorithm to do approximate belief state tracking on a DBN, and feed the marginals into various approximators for V, including SPOVA and a neural network, with mixed results.

[Thr99] uses PF to do approximate belief state tracking on a continuous-state model, and then uses a nearest neighbor function approximator for V; the distance metric is the KL distance between clouds of points that have been smoothed with a Gaussian kernel.

3.5 Heuristics based on the underlying MDP

We can compute Q for the underlying MDP, and combine this with the belief state in various heuristic ways. Here are some examples from [Cas98].

- Compute the most likely state, $x^* = \arg\max_x b(x)$, and define $\pi(b) = \pi^{MDP}(x^*)$ (the MLS approximation).
- Define $Q(b,a) = \sum_x b(x) Q^{MDP}(x,a)$ (the Q-MDP approximation).
- If the entropy of the belief state is below threshold, use one of the above heuristics, otherwise act to reduce the entropy (dual mode control).

GIB [Gin99], the world's best computer Bridge program, uses a very similar heuristic to Q-MDP: the belief state is estimated by sampling card distributions for each player which are consistent with $h_{1:t}$, and the underlying "MDP" is solved for each such distribution using a "double dummy" program.

The problem with these techniques is that they assume all uncertainty will "vanish" in the future, because the underlying MDP is fully observed. (In control theory, this is called "certainty equivalence"; see e.g., [SG00] for an example of (roughly) the MLS heuristic applied to a missile guidance problem, where the belief state is approximated using particle filtering.) Hence these techniques never perform information-gathering actions. This can be partially overcome by doing deeper lookahead: see Section 3.6.

3.6 Receeding horizon control

A simple idea is to look H steps into the future, and then pick the expected best action: see Figure 3. For example, looking two steps ahead, we define

$$\begin{aligned} Q(b_t, a_{t+1}, a_{t+2}) &= E_{x_t} E_{x_{t+1}} E_{e_{t+1}} E_{x_{t+2}} E_{e_{t+2}} \left[R(x_{t+1}, a_{t+1}) + \gamma R(x_{t+2}, a_{t+2}) + \gamma^2 V(b_2) \right] \\ &= \sum_{x_t} b_t(x_t) \sum_{x_{t+1}} P(x_{t+1}|x_t, a_{t+1}) [R(x_{t+1}, a_{t+1}) + \gamma \sum_{x_{t+2}} P(x_{t+2}|x_{t+1}, a_{t+2}) R(x_{t+2}, a_{t+2}) + \\ &\quad \gamma^2 \sum_{e_{t+1}} P(e_{t+1}|x_{t+1}, a_{t+1}) \sum_{e_{t+2}} P(e_{t+2}|x_{t+2}, a_{t+2}) \hat{V}_2(b_{t+2})] \end{aligned}$$

If we can compute this, we would pick $a^*_{t+1:t+2} = \arg\max_{a_{t+1:t+2}} Q(b_t, a_{t+1:t+2})$ by searching over action combinations. Note that we might only execute a_{t+1} and then replan, since the evidence e_{t+1} actually observed will update b_{t+1}.

[KMN99b] provide a theoretical analysis of this idea in the MDP case. They prove that at each node, it is sufficient to sample a number of following states which is independent of the complexity of the underlying MDP. This can be extended to belief-state MDPs by sampling observations instead of next states. This has complexity $O\left((|A|C)^H |S|^2\right)$, where C is the number of samples, $|A|$ is the number of actions, and $|S|$ is the size of the state space and H is the horizon (depth of lookahead). The $|S|^2$ can be reduced if the POMDP is factored (as in a DBN), and one does approximate belief state updating, e.g., using the BK algorithm [BK98]; error bounds on this combination of approximations are derived in [MS99]. Unfortunately, this method relies heavily discounting to keep the lookahead depth H tractable.

The lookahead controller just described does not learn a policy, and so does not become more effective with time. Furthermore, it estimates the values of the leaves as 0 (due to discounting). An alternative is to use an admissible heuristic (lower bound) for the value of the leaves (e.g., $V^{MDP}(b)$), and to update the values of the intermediate nodes

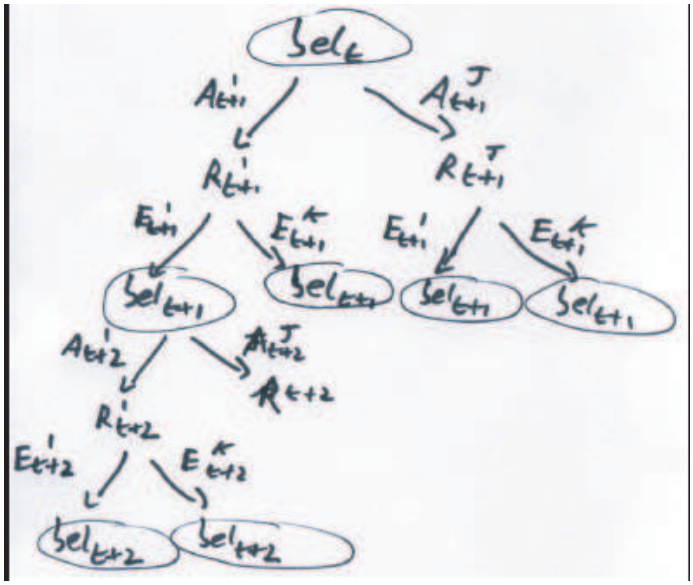

Figure 3: Forward search through belief state space. From the current belief state bel_t, we consider all possible discrete actions $A^1_{t+1}, \ldots, A^J_{t+1}$; each generates a reward R_{t+1} and a K possible percepts (evidence) $E^{1:K}_{t+1}$, resulting in a $J \times K$ possible new belief states bel_{t+1}.

as in Real Time Dynamic Programming [BBS95]. This is the approach taken in [GB98], who discretizes b, so V can be represented as a table. (See also [DNM98] for a way of combining look-ahead search with value functions for an MDP.)

4 Q-learning agents

4.1 Reactive agents

The simplest agent does not maintain any internal state, i.e., it is purely reactive/ memoryless. Equivalently, we define $S_t = Y_t$. For an environment with $|A|$ discrete actions and $|Y|$ discrete observations, there are $|A|^{|Y|}$ reactive policies. [Lit94] showed that the problem of finding an optimal deterministic reactive policy for a discrete POMDP is in general intractable, but for certain cases can be solved using branch-and-bound search. [LS98, Loc98] use Sarsa(λ) [SB98] to learn a deterministic reactive policy (Q function). (See also [PM98] for an analysis of what happens when RL techniques are applied to non-Markovian models.)

[SJJ94] showed that a deterministic reactive policy can do arbitrarily worse than a stochastic reactive policy, and in [JSJ94, WS98], they suggest a gradient ascent method for searching for reactive stochastic policies for discrete POMDPs, similar to the methods discussed in Section 5.2.3.

Reactive policies are plagued by "perceptual aliasing" (or "too little sensory data"), which is the problem that two states may appear the same but in fact are different. If the underlying states require different actions, this can lead to a loss of performance. Hence we now consider agents that use more than just the current observation.

4.2 Finite history windows

It is straightforward to use the last k observations as input to the policy. For discrete observations, these histories can be arranged into a suffix tree; at each leaf is the action that should be performed. The "Utile Suffix Memory" algorithm of [McC95] uses a suffix tree with variable depth leaves, as in a variable-length Markov model [RST96]. The "U-Tree" algorithm of [McC95] is a slight extension where the observation is treated as a vector, and different

branches can be created depending on the value of specific components of the history. This allows the policy to ignore irrelevant features in the input (the problem of "too much sensory data"). The result is a tree-structured Q function, where the tree is grown if a state (leaf in the tree) is not Markov wrt reward.

[HGM00] combines a hierarchical controller (formulated as a HAM [Par98]) with the "Nearest Sequence Memory" [McC95] approach. (NSM is like utile suffix memory, except a fixed length history is used.) This enables each level of the hierarchy to maintain its own perceptual history (define its own state space).

4.3 Recurrent neural nets

The problem with using a finite history is that important observations that were made in the past are forgotten. One approach is to use a recurrent neural network to maintain some long-term state [LM92, WL95].

5 Direct policy search

The most widely used approach to solving MDPs is an indirect one, namely computing Q (possibly by first solving for V in the case of a known model), and then using the greedy policy $\pi(s) = \arg\max_a Q(s, a)$. However, there are several problems with this is applied to POMDPs:

- Bellman's equation might not hold if the POMDP is not Markov wrt the agent's state-space, and hence it may not be possible to find a stable Q.
- Q-learning can fail to converge to a stable policy in the presence of function approximation, even for MDPs.
- Stochastic policies are better than deterministic ones in POMDPs.
- It may be hard to compute $\max_a Q(s, a)$ for continuous actions.
- The exploration-exploitation tradeoff.

Direct policy search does not suffer from any of these problems. There are two main issues: (1) how to represent the policy, and (2) how to learn the policy.

5.1 Policy representation

5.1.1 Reactive, stochastic controllers

It is common to represent a stochastic policy over discrete actions as follows:

$$\pi(s, i) = \frac{\exp \theta_i^T \phi(s)}{\exp \sum_k \theta_k^T \phi(s)} \tag{1}$$

where $\phi(s)$ is a feature vector for state s, and θ_i is the weight vector for action i. The feature vector and state-space is fixed in advance, but the weights can be learned using gradient descent.

For a stochastic policy over continuous actions, we can use a Gaussian as follows:

$$\pi(s, a) = N(a; \theta_1^T s, \theta_2)$$

where θ_2 controls the degree of exploratory behavior of the controller.

5.1.2 Finite state controllers

For some discrete POMDPs, the optimal controller has a finite number of states. One way to compute this FSM is to solve the belief state MDP, and convert the resulting policy graph into a FSM [KLC98]. A more direct way is to search directly in the space of FSMs. When the POMDP is unknown, [MPKK99] discusses gradient ascent. [PMK99] discusses a slight variation, where the state of the controller is represented in factored form (in terms of memory bits which can be individually addressed). When the POMDP is known, [MKKC99] discusses branch and bound and gradient ascent techniques, and [Han98] discusses policy iteration.

5.2 Learning the policy

5.2.1 Estimating $V(\pi)$ with a model

[KMN99a] suggest using a generative model of the POMDP to build a set of m trajectory trees, each of size $O(2^{H_\epsilon})$, to evaluate $V(\pi)$, where $H_\epsilon = \log_\gamma \frac{\epsilon(1-\gamma)}{2R_{max}}$ and ϵ is a user specifiable bound on the error in estimate of $V(\pi)$. Their main result is that only $O(VC(\Pi))$ trees need be built to evaluate $V(\pi)$ for all $\pi \in \Pi$, where $VC(\Pi)$ is the VC dimension of the policy class, i.e., the "sample complexity" is independent of the underlying POMDP. [NJ00] strengthens this result by building "trees" of size linear in H_ϵ, and allowing for continuous actions, assuming a known noisy functional environment model.

Given a way to compute $V(\pi)$, one can either enumerate all policies (for discrete state/action spaces), or use something like the Nelder-Mead simplex algorithm to find a good policy.

5.2.2 Estimating $\frac{\partial V(\pi)}{\partial \theta_\pi}$ with a model

If the controller is a differentiable function, we can try to use gradient ascent to learn it. The key problem is reliably estimating the gradient g. There are two main approaches: online and offline. In the online scenario, the POMDP is unknown, and the agent estimates g while interacting with the world, i.e., from a single sample path. We discuss this in the next section.

For offline learning, there are various techniques, depending on the kind of model we have. The weakest form of model is a generative model, in which case we can use conjugate gradient ascent, which will make repeated calls to an online gradient estimator [BWB99].

A completely different technique is presented in [NPK99]. They note that $V_{avg}(\pi) = d^\pi \cdot R$; they use approximate inference to compute the stationary distribution d^π (either BK or PF), and then take derivatives. For the BK algorithm, they can do this analytically, whereas for PF, they use an approximation based on importance sampling.

5.2.3 Estimating $\frac{\partial V(\pi)}{\partial \theta_\pi}$ without a model

REINFORCE [Wil92] was one of the first online gradient estimation algorithms. It was designed for immediate reinforcement problems. It updates the weights of the controller as follows:

$$\Delta\theta_{ij} = \alpha_{ij}(r - b_{ij})e_{ij}(s_t, a_t)$$

where $e_{ij}(s_t, a_t) = \frac{\partial}{\partial \theta_{ij}} \ln \pi(s_t, a_t)$, α_{ij} is a learning rate, r is the reinforcement, and b_{ij} is a reinforcement baseline (a fixed constant). He showed that $(E\Delta\theta)^T(E\nabla_\theta r) \geq 0$, and $(E\Delta\theta) = (E\nabla_\theta r)$ if $\alpha_{ij} = \alpha$ is fixed. Hence weights are updated, on average, in the direction of expected increasing reward. (The expectation is over any randomness in the environment and the policy.)

Williams suggested handling delayed reinforcement by running the above scheme until the end of an episode (e.g., when an absorbing state is reached), and repeating many times. The problem with this approach is that the long delays can introduce high variance into the estimate of the gradient. This can be overcome at the cost of a slight bias by introducing a pseudo discount factor [BB99, Mar98, KMK97]. This is equivalent to using an eligibility trace, and can be thought of as a way of letting future (discounted) rewards affect the present. The algorithm [KMK97] is as follows.

$$\begin{aligned} \delta_t &= r_t - b \\ z_t^a &= \lambda^a z_{t-1} + e(s_t, a_t) \\ \theta_{t+1} &= \theta_t + \alpha^a \delta_t z_t^a \end{aligned}$$

where z_t is the eligibility trace with discount factor λ. The OLPOMDP algorithm in [BWB99] is identical to this, except they use $b = 0$.

The above algorithm learns a controller (actor) directly, without means of a value function (critic). However, it is possible to concurrently learn a critic, and to use it as an adaptive reinforcement baseline. We can use an eligibility

trace for fitting V as well. In this case, the algorithm becomes

$$\begin{aligned}
\delta_t &= (r_t + \gamma V(s_{t+1})) - V(s_t) \\
z_t^c &= \gamma \lambda^c z_{t-1}^c + \nabla_w V(s_t) \\
w_{t+1} &= w_t + \alpha^c \delta_t z_t^c \\
z_t^a &= \lambda^a z_{t-1}^a + e(s_t, a_t) \\
\theta_{t+1} &= \theta_t + \alpha^a \delta_t z_t^a
\end{aligned}$$

where w are the parameters of the value function.

In the case of MDPs, there has been recent theoretical analysis of actor-critic algorithms that may prove useful to the POMDP case. Specifically, [KT99, SMSM99] show that a sufficient condition for the above algorithm to converge to a local optimum is that the critic learn a Q function which satisfies

$$\nabla_w Q(s, a) = \nabla_\theta \ln \pi(s, a) \tag{2}$$

If the actor uses a softmax π function,

$$\pi(s, a) = \frac{e^{\theta^T \phi(s,a)}}{\sum_b e^{\theta^T \phi(s,b)}} \tag{3}$$

then the compatibility requirement (Equation 2) gives

$$\nabla_w Q(s, a) = \phi_{sa} - \sum_b \pi(s, b) \phi_{sb}$$

so the Q function should have the form

$$Q_w^\theta(s, a) = w^T \left(\phi_{sa} - \sum_b \pi_\theta(s, b) \phi_{sb} \right) = w^T e_\theta(s, a)$$

The critic now uses SARSA(λ) to learn Q, and the actor no longer uses a trace. Also, the learning rate of the critic, α_c, must be much higher than that of the actor, so that the policy appears stationary as far as the critic is concerned. So the algorithm becomes

$$\begin{aligned}
\delta_t^c &= r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \\
z_t^c &= \gamma \lambda^c z_{t-1}^c + \nabla_w Q(s_t, a_t) \\
w_{t+1} &= w_t + \alpha^c \delta_t^c z_t^c \\
\theta_{t+1} &= \theta_t + \alpha^a Q_{w_t}^{\theta_t}(s_{t+1}, a_{t+1}) e_{\theta_t}(s_{t+1}, a_{t+1})
\end{aligned}$$

If the environment is partially observable, but the agent only uses observable features, the above result still holds.[4] The agent can also add (marginals of) the its belief state into its feature vector, since the belief state tracker conjoined with the environment is just one large MDP.

5.2.4 Evolutionary techniques

We can obviously use evolutionary algorithms to search the space of policies: see [MSG99] for details.

6 Learning an environment model

In the easiest case, the agent knows the underlying state space of the environment, as well as all the form of the functions $P(X_{t+1}|X_t, A_t)$, $P(Y_t|X_t, A_t)$, and $E(R_t|X_t, A_t)$. The functions are usually assumed to have simple

[4] Thanks to Andrew Ng for pointing this out.

parametric forms. In control theory, the transition/observation functions are often assumed to be linear-Gaussian, and the cost function quadratic. In AI, the transition/observation functions are often assumed to be discrete multinomial distributions, and the reward function can be an arbtitrary function of the discrete state/action. Discrete-state models may be represented in a factored form using a DBN [BDH99]. In any case, the parameters are assumed known.

A slightly more realistic case is when the parameters are unknown. For example, it might be known that the state space consists of the positions and velocities of the plane and missile, and that these objects are subject to Newton's laws, but certain parameters, such as the mass or type of the plane, or the noise variances, might be unknown. Furthermore, these parameters may change with time. This is the kind of problem studied in adaptive control theory.

Finally, the hardest case is where the agent knows "nothing" about the environment. Whether the agent should try to learn a model or not is a controversial topic in AI. In control theory, it is widely accepted that learning a model of the environment is useful; this is called system identification. If the model is assumed linear, there are well-established techniques; otherwise, people typically use neural networks. In the AI literature, there has been some work on learning discrete-state models of the environment. The "Utile Distinction Memory" algorithm of [McC95] combines the EM algorithm with a state-splitting approach to learn an HMM. [Sal99] learns a sigmoidal fully-interconnected DBN, using mean field inference and online gradient ascent. He applies it to a discrete driving task, but finds no advantages compared to learning an unfactored HMM. Both authors learn models to maximize the likelihood of the observations Y_t, which is not necessarily related to performance, although McCallum's state-splitting criterion at least is "utile", i.e., it only splits a state if it is not "Markov with respect to reward".

If we are prepared to make stronger assumptions about the kind of environment the agent will be embedded in, we can use more specialized representations, and get better performance. For example, a mobile robot in a 2 or 3-dimensional space (whether real or simulated) could benefit from the ability to represent a map of the world. A robot designed to deliver mail would benefit from the ability to represent the appearance and typical locations of people and objects. And so on. It is clear that to make reinforcement learning work on any real domain, a lot of structure will have to be built in by hand.

References

[BB99] J. Baxter and P. Bartlett. Direct gradient-based reinforcement learning: I. gradient estimation algorithms. Technical report, Dept. Comp. Sci., Australian Natl. Univ., 1999.

[BBS95] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynammic programming. *Artificial Intelligence Journal*, 72:81–138, 1995.

[BDH99] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *J. of AI Research*, 1999.

[BK98] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. of the Conf. on Uncertainty in AI*, 1998.

[BP96] C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *AAAI*, 1996.

[Bra97] R. Brafman. A heuristic variable grid solution method for POMDPs. In *AAAI*, 1997.

[BWB99] J. Baxter, L. Weaver, and P. Bartlett. Direct gradient-based reinforcement learning: II. gradient ascent algorithms and experiments. Technical report, Dept. Comp. Sci., Australian Natl. Univ., 1999.

[Cas98] A. Cassandra. *Exact and approximate algorithms for partially observable Markov decision processes*. PhD thesis, U. Brown, 1998.

[DdFG01] A. Doucet, N. de Freitas, and N. J. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2001.

[DNM98] S. Davies, A. Ng, and A. Moore. Applying online-search to reinforcement learning. In *AAAI*, 1998.

[DW91] T. Dean and M. Wellman. *Planning and Control*. Morgan Kaufmann, 1991.

[GB98] H. Geffner and B. Bonet. Solving large POMDPs using real time dynamic programming. In *Fall AAAI Symp. on POMDPs*, 1998.

[Gin99] M. Ginsberg. Gib: Steps toward an expert-level bridge-playing program. In *Intl. Joint Conf. on AI*, 1999.

[Han98] E. Hansen. Solving POMDPs by searching in policy space. In *Proc. of the Conf. on Uncertainty in AI*, 1998.

[Hau00] M. Hauskrecht. Value-function approximations for partially observable markov decision processes. *J. of AI Research*, 13:33–94, 2000.

[HGM00] N. Hernandez-Gardiol and S. Mahadevan. Hierarchical memory-based reinforcement learning. In *NIPS-13*, 2000.

[JSJ94] T. Jaakkola, S. Singh, and M. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In *NIPS-7*, 1994.

[KLC98] L. P. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

[KMK97] H. Kimura, K. Miyazaki, and S. Kobayashi. Reinforcement learning in POMDPs with function approximation. In *Intl. Conf. on Machine Learning*, 1997.

[KMN99a] M. Kearns, Y. Mansour, and A. Ng. Approximate planning in large POMDPs via reusable trajectories. In *NIPS-12*, 1999.

[KMN99b] M. Kearns, Y. Mansour, and A. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *Intl. Joint Conf. on AI*, 1999.

[KT99] V. Konda and J. Tsitiklis. Actor-critic algorithms. In *NIPS-12*, 1999.

[Lit94] M. Littman. Memoryless policies: theoretical limitations and practical results. In *Proc. of the Conf. on Simulation of Adaptive Behavior*, 1994.

[LM92] L-J. Lin and T. Mitchell. Memory approaches to reinforcment learning in non-Markovian domains. Technical Report CMU-CS-92-138, CMU, 1992.

[Loc98] J. Loch. The effect of eligibility traces on finding optimal memoryless policies in partially observable Markov decision processes. In *NIPS-11*, 1998.

[LS98] J. Loch and S. Singh. Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *Intl. Conf. on Machine Learning*, 1998.

[Mar98] P. Marbach. *Simulation-Based Optimization of Markov Decision Processes*. PhD thesis, MIT LIDS, 1998.

[McC95] A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Univ. Rochester, 1995.

[MKKC99] N. Meuleau, K-E. Kim, L. Kaelbling, and A. Cassandra. Solving POMDPs by searching the space of finite policies. In *Proc. of the Conf. on Uncertainty in AI*, 1999.

[MPKK99] N. Meuleau, L. Peshkin, K-E. Kim, and L. Kaelbling. Learning finite-state controlers for partially observable environments. In *Proc. of the Conf. on Uncertainty in AI*, 1999.

[MS99] D. McAllester and S. Singh. Approximate planning for factored POMDPs using belief state simplification. In *Proc. of the Conf. on Uncertainty in AI*, 1999.

[MSG99] D. Moriarty, A. Schultz, and J. Grefenstette. Evolutionary algorithms for reinforcement learning. *J. of AI Research*, 11:241–276, 1999.

[NHR99] A. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Intl. Conf. on Machine Learning*, 1999.

[NJ00] A. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proc. of the Conf. on Uncertainty in AI*, 2000.

[NPK99] A. Ng, R. Parr, and D. Koller. Policy search via density estimation. In *NIPS-12*, 1999.

[Par98] R. Parr. *Hierarchical Control and Learning for Markov Decision Processes*. PhD thesis, U. C. Berkeley, 1998.

[PB00] P. Poupart and C. Boutilier. Value-directed belief state approximation for POMDPs. In *Proc. of the Conf. on Uncertainty in AI*, 2000.

[PM98] M. Pendrith and M. McGarity. An analysis of direct reinforcement learning in non-Markovian domains. In *Intl. Conf. on Machine Learning*, 1998.

[PMK99] L. Peshkin, N. Meuleau, and L. Kaelbling. Learning policies with external memory. In *Intl. Conf. on Machine Learning*, 1999.

[PR95] R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Intl. Joint Conf. on AI*, 1995.

[RPK99] A. Rodriguez, R. Parr, and D. Koller. Reinforcement learning using approximate belief states. In *NIPS-12*, 1999.

[RST96] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25, 1996.

[Sal99] B. Sallans. Learning factored representations for partially observable Markov decision processes. In *NIPS-12*, 1999.

[SB98] R. Sutton and A. Barto. *Reinforcment Learning: An Introduction*. MIT Press, 1998.

[SG00] D. Salmond and N. Gordon. Particles and mixtures for tracking and guidance. In A. Doucet, N. de Freitas, and N. J. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2000.

[SJJ94] S. Singh, T. Jaakkola, and M. Jordan. Learning without state-estimation in partially observable Markov decision processes. In *Intl. Conf. on Machine Learning*, 1994.

[SMSM99] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS-12*, 1999.

[SPS99] R.S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[Thr99] S. Thrun. Monte Carlo POMDPs. In *NIPS-12*, 1999.

[Wil92] R. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[WL95] S. Whitehead and L-J. Lin. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73:271–306, 1995.

[WS98] J. Williams and S. Singh. Experimental results on learning stochastic memoryless policies for partially observable Markov decision processes. In *NIPS-11*, 1998.