Reinforcement Learning for Problems with Hidden State

Samuel W. Hasinoff
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
hasinoff@cs.toronto.edu

September 9, 2003

Abstract

In this paper, we describe how techniques from reinforcement learning might be used to approach the problem of acting under uncertainty. We start by introducing the theory of partially observable Markov decision processes (POMDPs) to describe what we call hidden state problems. After a brief review of other POMDP solution techniques, we motivate reinforcement learning by considering an agent with no previous knowledge of the environment model. We describe two major groups of reinforcement learning techniques: those the learn a value function over states of world, and those that search in the space of policies directly. Finally, we discuss the general problems with these methods, and suggest promising avenues for future research.

1 Introduction

Consider a robot designed to perform basic housekeeping tasks in an office building. We would like the robot to make informed decisions which account explicitly for the uncertainty in the world. In interacting with the environment, the robot may take actions with uncertain effects. In attempting to vacuum a section of carpet, for example, the robot may encounter an undetected obstacle, or its vacuum actuators may fail. Moreover, the observations made by the robot may also be uncertain. Range-finders for indicating the presence of walls may give inaccurate readings, and worse yet, the robot may have difficulty distinguishing a certain corner of an office from a similar corner in another office down the hall. The owner of the robot might impose severe penalties for the robot letting its battery run down, or for falling down a flight of stairs, whereas the robot might be rewarded such behaviour as completing housekeeping without annoying the office workers.

In general, we have the problem of an agent acting under uncertainty to maximize reward. Problems of this sort are central to artificial intelligence, control theory, and operations research. In the most basic formulation, only the uncertainty in actions is modelled. However, if uncertainty is also allowed in the observations, the problem becomes much more difficult, since the true state of the environment is hidden.

But problems with hidden state are everywhere. Indeed, most sequential decision problems can be made to fit this rather general form. We would like to develop good approaches for these problems that are grounded with solid theoretical foundations. After introducing a formalism for describing these kinds of problems, and a brief review of other solution techniques, we focus our attention on the case where the environment model is unknown to the agent. Under these circumstances, the agent must apply some kind of reinforcement learning technique, interacting directly with the environment to learn how best to act under multiple forms of uncertainty.

2 POMDP Review

The partially observable Markov decision process (POMDP) is a powerful formalism for representing sequential decision problems for agents that must act under uncertainty. At each discrete time step, the agent receives some stochastic observation related to the state of the environment, as well as a special reward signal. Based on this information, the agent can execute actions to stochastically change the state of the environment. The goal of the agent is then to maximize the overall (and typically time-discounted) reward.

Following the treatment of (Kaelbling, Littman and Cassandra, 1998), a POMDP can be formally described as a tuple $\langle S, A, T, R, O, \Omega \rangle$, where

- S is a finite set of states of the environment;
- A is a finite set of actions;
- $T : S \times A \to \Delta(S)$ is the state-transition function, giving a distribution over states of the environment, given a starting state and an action performed by the agent;
- $R : S \times A \to \mathcal{R}$ is the reward function, giving a real-valued expected immediate reward, given a starting state and an action performed by the agent;
- Ω is a finite set of observations the agent can experience; and
- $O : S \times A \to \Delta(\Omega)$ is the observation function, giving a distribution over possible observations, given a starting state and an action performed by the agent.

Note that the sub-tuple $\langle S, A, T, R \rangle$ represents the underlying MDP. If the observation function were to give the true (hidden) state of the environment with perfect certainty, the problem reduces to a fully observable MDP. In general, this is not the case. The same observation may occur in more that one state of the environment, and these states may require different actions.

We can also consider the internal state maintained by the agent. Let Y refer to the finite set of all possible internal agent states. To illustrate, the situation where $Y_t = O_t$ describes a memoryless agent, whose next action depends only on the current observation. Figure 1 shows the structure of a POMDP in terms of the interaction between the agent and the environment, using a temporal Bayesian network.

For a more complete introduction to POMDPs, particularly exact solution techniques, see the well-written review paper by Kaelbling, Littman and Cassandra (1998).

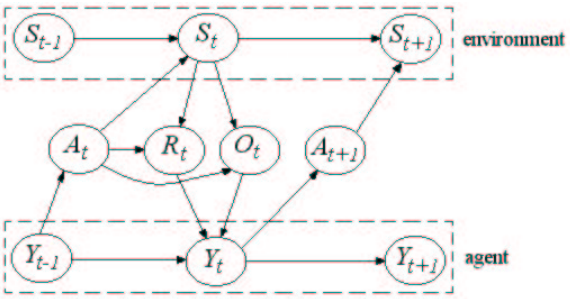

Figure 1: POMDP represented as a temporal Bayesian network. The agent maintains some kind of (finite) internal state, and interacts with the environment based on previous observations and rewards. All of the transitions shown in the diagram can be stochastic.

3 Environment model

The problem faced by the agent of acting well under uncertainty can be considerably more or less difficult based on what sort of model of the environment is available. One slightly unrealistic possibility is that the model of the environment is completely known. At the other extreme, nothing whatsoever might be known about the environment. In this case, the only information available to the agent is information obtained from direct experience. Somewhere in between, a generative model of the environment might be available, allowing the agent to perform sampling or simulate experience with the environment off-line.

Of course, any knowledge about the model of the environment can be ignored. For example, for computational reasons, the agent might consider using sampling techniques even if the full model of the environment were at its disposal.

3.1 Known environment model

In the most straightforward case, the POMDP is fully specified, meaning that all of the dynamics are known in advance. In other words, the transition function, the reward function, and the observation function are all available to the agent.

When the environment model is known, the standard approach is for the agent to compute the belief state b (the probability distribution over state space S) and maintain this as its internal state. Since the current belief state completely summarizes the initial belief state and all previous experience with the environment, the belief state is said to comprise a sufficient statistic.

Every discrete POMDP can be reformulated as a continuous-space MDP whose states are belief states. In fact, it can be shown that the value function of this new MDP is piecewise linear convex, so that even though state space is continuous, finding an exact solution (or an arbitrarily optimal solution for the infinite-horizon case) is always possible. However, while possible, finding an exact solution to a POMDP is highly intractable, and in the worst case is doubly exponential in the horizon time. The best exact algorithms cannot handle more than a few dozen states, so some form of approximation is typically used.

One approach to approximately solving POMDPs is to compute a (state-action) value function for the underlying MDP and combine this with the belief state using various heuristics. Cassandra (1998) describes several examples of this technique. The simplest such heuristic is the MLS (most likely state) approximation, in which the underlying MDP is used to select actions based on the current state with highest probability (this is akin to particle filtering in a discrete setting). Another such heuristic is the Q-MDP approximation, in which the value function for the belief-state MDP is estimated from the value function for the underlying MDP weighted according to the belief state.

Another approach to approximately solving POMDPs is to maintain an exact belief state, but approximate the (piecewise linear convex) value function of the belief-state MDP. Parr and Russell (1995) suggest a smooth, differentiable approximation to the value function with their SPOVA algorithm. A different strategy along this vein is to discretize the value function and interpolate.

It is also possible to do the opposite, maintaining an exact value function for the belief-state MDP, but approximating the belief state. The Boyen-Koller algorithm (Boyen and Koller, 1998) does this for an environment model specified compactly as a dynamic Bayesian network (DBN), using sampling to perform approximate belief state updating.

3.2 Generative model of the environment

Sometimes, the full dynamics of the POMDP are unknown, but a generative model or accurate simulation of the environment is available. This situation permits learning off-line, without directly interacting with the environment.

Particle filtering and other sampling approaches for estimating the belief state are reviewed in (Doucet, Godsill, and Andrieu, 2000). These techniques are particularly appropriate for continuous-valued POMDPs.

Receding horizon control (Kearns, Mansour, and Ng, 1999a) is a sampling technique in which a lookahead tree of fixed depth H is built. It can be shown that the number of states that must be sampled is independent of the complexity of the underlying MDP. However, this method never actually learns the policy (and so does not become more effective over time), and relies heavily on a high discount rate to keep the lookahead depth tractable.

Finally, it is possible to combine sampling from the generative model with a direct search in the space of policies. One can estimate the value of a policy by sampling a number of trajectories (Kearns, Mansour and Ng, 1999b) and then choose a good policy by enumeration or something similar to the simplex algorithm. If the value function is differentiable with respect to the (parameterized) policy then the gradient can be estimated by sampling. In this case, a variety of numerical optimization methods, such as conjugate gradient methods, can be used to carry out policy improvement.

3.3 No environment model

Without a model of the environment, the only method of gaining truly new experience is to directly interact with environment. This sort of situation is perhaps the main motivation for reinforcement learning (RL). Thus, for the remainder of this paper, we will assume that no model of the environment is available to the agent.

There is some debate about whether it is worthwhile for the agent to construct its own internal model of the environment, for example, by gathering statistics about transitions and observations. Building a good model for POMDPs is more difficult than in the fully observable case, since the transition probabilities might depend on the policy being executed. So, while creating a model can give plausible results, there are few theoretical guarantees about the effectiveness of this strategy. Some of the reinforcement learning methods we will describe do build models.

4 Reinforcement Learning Methods

For the purposes of this paper, we take the notion of reinforcement learning broadly, to mean any trial-and-error interaction with the environment with the goal of improving some (possibly time-discounted) reward signal. We assume that no model of the environment is given, and that learning is unsupervised, meaning that no training examples are given.

Reinforcement learning methods can be roughly separated into two groups: those that learn the policy indirectly, by constructing a value function over states and actions, and those that search the space of policies directly. For a good introduction to the former see (Sutton and Barto, 1998; Kaelbling, Littman, and Moore, 1996). The latter group is much more diverse.

4.1 Learning the value function

In the fully observable MDP setting, the most popular reinforcement learning techniques involve gradually learning $Q(s, a)$, the value function over state-action space, through statistical techniques and dynamic programming. As the Q-values become closer to exact, performing the greedy policy becomes closer to optimal.

Using this approach for POMDPs presents a number of complications. In the presence of hidden state, the greedy policy can be arbitrarily worse than optimal, even given perfect knowledge of the value function. In fact, many of the methods that learn the value function can fail to converge in the presence of function approximation. Furthermore, even the act of choosing the greedy policy can be difficult in the presence of continuous actions.

4.1.1 Memoryless policies

The simplest approach along these lines is to ignore the complications and apply Q-learning (Watkins, 1989) or some related technique to the POMDP directly. Since the agent maintains no internal state regarding the history of its observations, the policies (mapping from observations to actions) generated by this method are known as memoryless, or reactive. In the best case, if the POMDP is close to Markov and has a good memoryless policy, we might hope to find it using this technique.

Some of the theoretical limitations of using memoryless policies are explored by Littman (1994). The problem of finding the optimal memoryless policy for a POMDP is shown to be NP-Hard, but is basically amenable to branch-and-bound techniques. In addition, simple examples can be constructed which cause Q-learning to oscillate.

Loch and Singh (1998) remark that reinforcement learning techniques with eligibility traces seem to perform empirically much better than Q-learning on POMDPs with good memoryless policies. Using Sarsa(λ), they obtain optimal solutions for some of the (small) problems found in the POMDP literature, and note that the theoretical analysis in (Littman, 1994) ignores eligibility traces altogether.

Interestingly, deterministic memoryless policies can be shown to be arbitrarily worse than stochastic ones (Singh, Jaakkola and Jordan, 1994). However, most methods avoid searching the continuous space of stochastic memoryless policies for computational reasons. Figure 2 shows a simple example of this phenomenon.

It is important to realize that even the best memoryless policies can have poor performance, particularly if explicit information gathering is required of the agent. However, there exists nonetheless an important class of POMDPs for which, although there may be poor observability, the best memoryless policy gives near-optimal return.

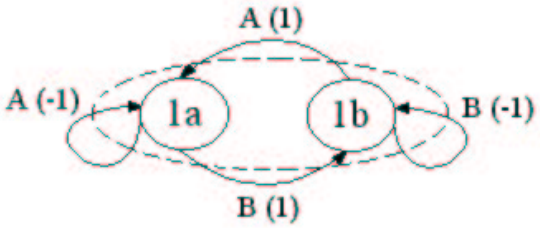

Figure 2: Two state POMDP for which the optimal memoryless policy is stochastic. The large dashed ellipse indicates that both states are aliased to the same observation. Transitions are labeled with actions, with immediate reward given in parentheses. The best a deterministic memoryless policy can achieve is a reward of 1, followed by an infinite sequence of -1's. By contrast, a stochastic policy choosing actions A and B with equal probability will have an expected reward of 0 at each time step.

4.1.2 Using memory to maintain an internal state

Many researchers have noted that a fundamental problem in working with POMDPs is perceptual aliasing, the situation in which several states of the system are aliased to the same observation. This problem is variously referred to as the hidden state problem, or incomplete perception. Figure 3 gives a concrete example.

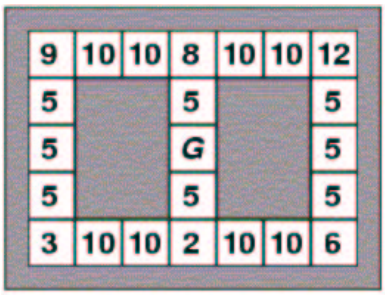

Figure 3: Example of perceptual aliasing in a simple maze environment. The agent is not aware of its true state in the environment, but instead receives observations that describe whether there are walls immediately in each of the four cardinal directions. Note that not all of the 16 possible observations actually occur in this maze. The goal state, which is assigned a positive reward, is labelled G.

Under these conditions, memoryless policies are clearly insufficient in general, since they can associate at most one action with each observation. Whitehead and Ballard (1991) offer a trivial solution with their Lion algorithm, which is simply to avoid passing through aliased states. A better idea is to introduce some form of memory, so that the agent can attempt to use its past experiences

to disambiguate aliased states and act appropriately.

Simple memory-based approaches In the literature, various simple-minded *ad hoc* approaches to using memory for handling perceptual aliasing have been suggested. Loch and Singh (1998) expand the state space (exponentially) to include the previous k observations. Along the same lines, Littman (1994) augments state space with a single bit, and provides new actions to set and reset the bit. While these ideas clearly will not scale beyond the small problems at which they are targeted, they do provide inspiration for more effective methods.

Nearest sequence memory McCallum (1995a) proposes a very straightforward method called nearest sequence memory (NSM). This method operates by recording the history of experiences, or in other words, the actions taken and the resulting observations and rewards. For computational reasons, the history can be limited to containing some reasonably large number of the most recent experiences.

Given a history of experiences, a distance metric is applied to determine the k previous states that are closest to the current state. The metric used by NSM is the number of matching preceding experiences, with the rationale that states with histories similar to the current state will more likely represent the true (hidden) state of the agent. Note that this distance metric could possibly be extended beyond exact history matching, in order to handle stochastic rewards or continuous observation space.

Next, the k nearest neighbours are used to obtain Q-values, by averaging the expected future reward values for each action. The action with the highest calculated value is executed, and the new resulting experience is added to the history of the agent. Finally, the standard Q-learning update rule is invoked for those states that led the agent to choose the action that it did.

While NSM embodies perhaps the most basic (but sensible) memory-based approach to reinforcement learning, it exhibits surprisingly good performance. Good policies were discovered quickly for simple maze environments with a high degree of perceptual aliasing, as well as for other examples from the POMDP research literature. Here the memory serves the agent as both a non-stationary representation of the policy and a simple model of the environment. In a weak sense there is also a connection between NSM and other sampling techniques.

NSM is at heart a heuristic method, and as such does not guarantee any particular theoretically justified level of solution quality. As might be expected, NSM does not handle noise well, since there is no explicit mechanism for separating noise from the structure of the problem. Furthermore, the NSM approach is really most appropriate for modeling short-term memory. The method should be expected to have difficulty if it is required to correlate an important observation from the distant past with the current state. In general, NSM will probably not scale very well to large problems, although perhaps some leverage could be gained in combination with a hierarchical approach.

Utile distinction memory Chrisman (1992) and McCallum (1993) describe similar approaches for learning POMDPs that involve building a kind of probabilistic finite state machine. The states of the machine are split based on batched analysis of statistics gathered over many steps, and the current state of the finite state machine acts as memory to help distinguish perceptually aliased states.

The utile distinction memory (UDM) algorithm of (McCallum, 1993) has the interesting feature that the finite state machine is only expanded when doing so will increase the ability of the agent to predict reward. In this sense, the perceptual distinctions made by the state machine are utile. To achieve this end, a robust statistical test helps distinguish between genuine variation in the predicted reward and noise.

While UDM contains some clever ideas, it is not a practical approach for two reasons. First, the algorithm operates within a certainty-equivalence framework, in which the agent alternates between periods of gathering statistics for the current model and modifying the model based on these statistics. This approach is notoriously slow, since many steps must be taken to ensure statistically significance, and very few changes to the model can typically be made at the end of each iteration. The second problem with UDM is the difficulty it has in discovering the utility of memories longer than one time step. This is to be expected, since the statistical test only examines the predictive benefit of making a single additional split.

Utile suffix memory In an effort to combine the best features of nearest sequence memory and utile distinction memory, McCallum (1995b) introduces utile suffix memory (USM). This approach records the history of experiences in the same straightforward manner as nearest sequence memory, but also organizes these experiences in the leaves of a tree. The tree is known as a suffix tree, because the root node splits on the current observation, and deeper nodes correspond to experiences further in the past. Leaves can occur at different depths, as in a variable-length Markov model, so that deeper branches use more memory to distinguish states finely, and shallower branches use less memory and make broader distinctions.

The suffix tree is also augmented with fringe nodes, extending to some depth below the leaves, in order to allow the testing of additional distinctions. The Kolmogorov-Smirnov test, which is similar in spirit to a chi-squared test, is used to check if two distributions are significantly different. Fringe nodes are compared to their ancestor leaves for differences. If a fringe node is judged significantly different from its ancestor leaf, it has additional power to predict reward, and is thus promoted to a full-fledged leaf. Like in UDM, the only distinctions made are those with proven utile value. Additional fringe is extended below the new leaf, and previous experience from the old ancestor leaf can be properly partitioned by looking further back in time. For efficiency reasons, this analysis might only be performed after a certain number of new experiences.

To choose the best action to perform, the suffix tree is used to determine the

leaf node that corresponds to the most recent observations and actions. Among the experiences stored at the leaf, the one with the highest Q-value is chosen, and its related action is executed. The resulting experience is added to the history of the agent and also associated with the leaf.

McCallum notes that the regular Q-learning rule can applied like in NSM, but instead proposes a different model-based approach for updating the Q-values. The transition and reward functions can be estimated directly from recorded experience, giving an approximate model of the environment. This model can then be used to perform one complete sweep of value iteration. If computational limitations prevent this, an approach like prioritized sweeping (Moore and Atkenson, 1993) is appropriate, in which only a certain number of states, those predicted to be the most influential in modifying the value function, are backed up.

USM is empirically shown to perform better than any of the previously described memory-based approaches. It takes consistently less steps to converge, finds good quality solutions, and is just as fast in terms of computational time. Although noise in the actions and reward function is handled explicitly, perceptual noise is not, in contrast to other POMDP solution methods. Not surprisingly then, USM gives solutions with no theoretical guarantees on optimality. Another problem with the method involves choosing the size of the fringe. If the fringe is too large, then statistical testing will unnecessarily dominate the computation time. On the other hand, if the fringe is too small, the method will have a similar problem to UDM, in that the algorithm could have difficulty discovering the utility of memories longer than the depth of the fringe.

Recurrent-Q Lin and Mitchell (1992) describe a neural network approach to learning Q-values that they call Recurrent-Q. Neural networks can be characterized as recurrent if there are backward looping connections from the hidden units to the input layer. In this way, previous inputs become relevant to the neural network, and features from the history can be learned and stored in the structure of the network. Recurrent-Q has had success with simple problems, but more work needs to be done to explore how one might scale this approach to larger problems and avoid settling on local optima.

4.1.3 Hierarchical reinforcement learning

Hierarchical reinforcement learning methods hold tremendous potential for gaining computational leverage in order to solve large-scale decision problems. Different levels of representation are appropriate in different places, so it is natural to decompose the state space for the purpose of abstracting away unnecessary detail. For example, whether the agent has the goal of going to the coffee shop across the street or downtown Beijing, the policy for leaving the current room should be exactly the same.

Early work on hierarchical methods (Kaelbling, 1993) involved the programmer manually partitioning state space, and setting appropriate milestones in

each region in advance. High level information was then used to navigate from milestone to milestone along the shortest path to the region containing the goal.

Wiering and Schmidhuber (1998) propose HQ-learning, a hierarchical extension of Q-learning in which POMDPs are decomposed into a (fixed) number of memoryless policies. This work is related to, but more general than, other forms of multiple agent Q-learning such as Feudal Q–learning and W-learning. HQ-learning involves learning both the single-layer decomposition and the optimal memoryless policies simultaneously. Another interesting feature of the method is that learned memoryless policies can be reused in different parts of state space. Good results are shown on partially observable maze environments with a relatively large number of states. One of the main problems with the method is managing the transfer of control between sub-policies in the presence of noise.

Different frameworks have recently been suggested for hierarchical reinforcement learning in which the hierarchies are given in advance by the programmer, based on domain knowledge. The MAXQ framework (Dietterich, 1998) involves constructing a hierarchy of subtasks, and also decomposing the value function. This method is formally analysed as a semi-MDP, in which actions (subtasks) can take variable amounts of time, and convergence properties are proven. Parr and Russell (1998) take a different approach with their hierarchical abstraction of machines (HAM) framework. HAM involves constructing hierarchies of finite state controllers that can call each other like procedure calls. In a fully observable MDP setting, the learned policy can be shown to be optimal with respect to constraints imposed by the controllers in the hierarchy.

Hernandez-Gardiol and Mahadevan (2000) combine the HAM framework with the nearest sequence memory and utile suffix memory suggested by McCallum for a complex simulated robot task. They illustrate the great advantage of hierarchical approaches over learning in flat primitive space, and also suggest that it is worthwhile to introduce memory-based approaches at all different levels of the hierarchy. Unifying memory-based and hierarchical approaches to reinforcement learning for problems with hidden state seems like a very promising area for future research.

4.2 Direct policy search

The second major group of reinforcement learning methods are those that search the space of policies directly. These methods do not face the same host of problems as methods that operate by learning the value function. However, direct policy search is not without its own problems. Unless suitable constraints can be imposed, even the space of small finite policies is enormous, and these methods can be exceedingly slow. Furthermore, although many of these methods come with strong theoretical guarantees on convergence, none give any guarantees on solution quality. Direct policy search is beleaguered with the problem of becoming trapped on low-quality local optima.

4.2.1 Evolutionary algorithms

The evolutionary approach, as reviewed in (Moriarty, Schultz, and Grefenstette, 1999), is often overlooked by the rest of the reinforcement learning community. It is true that evolutionary algorithms comprise not nearly as cohesive a body of research as the more standard reinforcement learning methods. What evolutionary algorithms do have in common though is roots in function optimization, inspiration from biological systems, and the practice of assessing (typically parameterized) policies directly. Broadly speaking, evolutionary algorithms are slower, take less memory, and do not handle rarely visited states very well.

Classifier systems Classifier systems, which evolve symbolic rules mapping from input values to actions, were the first evolutionary algorithms ever developed. Dorigo and Bersini (1994) review the strong connection between techniques for credit assignment in classifier systems and in Q-learning. There have been few successful applications of classifier systems, but the framework is still interesting because of the way it unifies ideas from different branches of reinforcement learning

Genetic algorithms Long employed as a heuristic method for function optimization, genetic algorithms (Goldberg, 1989) deserve some attention as a policy search method for solving POMDPs. Genetic algorithms operate by maintaining a population of policies, where the fitness of each policy is judged directly from the reward obtained by interacting with the environment. At each generation, policies are randomly perturbed (mutated), and spliced together to form new combinations (crossed over). Only the policies with the highest fitness are allowed to move on to the subsequent generation.

Genetic algorithms are quite slow, since in some sense, at every iteration, a new tournament needs to be run between all policies in the population. This method is really better suited to learning episodic, goal-based tasks, where the model of the environment is available so that learning can be performed off-line. Genetic algorithms do tend to give good results in the long run, and they work well with parameterized representations of policies and in highly non-smooth policy spaces.

Genetic programming The idea behind genetic programming is to evolve actual computer programs to represent the policy. Indexed memory (Teller, 1994) is a method of augmenting basic genetic programming with a finite amount of memory as well as load and store instructions. In order to go beyond simple memoryless policies, some method like this is required. In fact, the class of programs that can be evolved with indexed memory can be shown to be Turing complete. This flexibility is paid for dearly in terms of an extremely slow learning rate, and this method is currently only practical for small problems.

Schmidhuber (1997) presents a novel algorithm called success-story algorithm (SSA) which extends a form of genetic programming known as Levin search. It is possible to show that Levin search is asymptotically optimal for a

wide class of problems, but this method is only recently finding practical applications. SSA develops a restricted framework for learning how to learn, in which previously learned pieces of program can be adapted to new circumstances, and the utility of attempting to do this is periodically estimated. Schmidhuber has shown good results using SSA on very large problems (over 1018 states), and it seems that this type of approach bears much closer inspection.

4.2.2 Gradient ascent methods

The second group of policy search methods require the value of the policy to be a differentiable function. If this condition holds, it is reasonable to estimate the gradient and use this to perform some variant of gradient ascent. Note that in the absence of an environment model, this means estimating the gradient online through direct interaction with environment, and following a single trajectory in policy space. These methods in particular have problems with getting trapped on poor quality local optima, since they consist of a form of stochastic local search.

The REINFORCE algorithm (Williams, 1992) was one of the first applications of this idea, but it was slow, operated only on memoryless policies, and its method for gradient estimation had high variance. Several authors (Sutton, McAllester, Singh, and Mansour, 1999; Baird and Moore, 1998) have since developed a better method for estimating the gradient. Even more importantly, their new method is shown to converge even in the presence of (reasonable) function approximation.

Memoryless stochastic policies Another early application of gradient search was the algorithm proposed in (Singh, Jaakkola and Jordan, 1994) for finding optimal memoryless stochastic policies. The algorithm uses a Monte Carlo approach for policy evaluation, and does gradient ascent for policy improvement. Although the space of stochastic policies is continuous, the algorithm is computationally tractable, and is shown guaranteed to converge to a local optimum.

Finite state controllers The solution to a particular POMDP can be approximated by finding the best policy representable as a finite state controller of a given size. This technique is intuitively effective because many real-world problems have sufficient structure that near-optimal solutions can be described in a highly compact form such as a finite state controller.

Gradient ascent techniques can be applied to finding locally optimal stochastic policies represented as finite state controllers (Meuleau, Kim, Kaelbling, and Cassandra, 1999). Good results were shown on a difficult pole-balancing task, but it was noted the method is much slower than standard reinforcement learning techniques for situations in which a good memoryless policy existed. Moreover, as the finite state controller was allowed to grow in size, the solution quality was shown to improve, but the running time of algorithm increased significantly. Unfortunately, it was difficult to predict when learning would level off

as the size of the finite state controller was increased, since the solution quality appeared to improve in jumps.

A related technique uses gradient ascent to learn policies with a finite amount of external memory (Peshkin, Meuleau, and Kaelbling, 1999). The space of actions is augmented with new actions to toggle each of the bits, so that changing the overall state of the memory may take multiple steps. This approach is reminiscent of naïve memory-based methods in Littman (1994).

In fact, the framework of policies as finite state controllers is general enough to include a variety of methods that we have already seen. Policies with external memory, HQ-learning, and the finite-horizon memory techniques due to McCallum can all be thought of as imposing special structural constraints on the larger set of all possible finite state controllers.

5 Discussion

Reinforcement learning is a rich body of research that gives us many useful techniques to attack the problem of acting under uncertainty. However, there are still fundamental problems with the approach.

The trade-off between exploiting the best known policy and performing further exploration of the environment is an important issue in reinforcement learning, but one that is very poorly understood for POMDPs. Furthermore, some formulations assume the learning problem is goal-based and episodic (in other words, a planning problem). Instead, we would prefer general solutions for ongoing, infinite-horizon problems with a more flexible reward structure. An additional problem with reinforcement learning methods is that they are typically very slow to converge on a good solution. One related issue is that most reinforcement learning researchers make the restrictive assumption that the agent always starts from a state of zero knowledge. In reality there is often important domain knowledge and expert advice that could be incorporated from the outset. Without an environment model or additional guidance from the programmer, the agent may literally have to keep falling off the edge of a cliff in order to learn that this is bad behaviour.

In the past ten years, great strides have been made by the POMDP community, but effectively solving large, real-world problems remains elusive. The approaches that look the most promising for the future are hierarchical and memory-based approaches. There is a clear advantage to using factored models of state space and reward, and arranging decision-making hierarchically. However, new frameworks need to be developed to do this even more flexibly. Memory-based methods are important for disambiguating hidden state, and recent research suggests that an agent can benefit from memory at many different levels. We have seen the usefulness of both general and structurally constrained finite state controllers for representing policies, but in even more generality we would like to learn (compact) programs for carrying out polices.

While the POMDP formalism is highly general, there are still a variety of desirable extensions to consider. We would like to be able to work in mixed

spaces (with both continuous and discrete components) of actions and observations. Moreover, we might consider extending POMDPs to better cope with non-stationary environments. This problem has proven disappointingly difficult for reinforcement learning, and so far only slowly drifting environments can be handled. Multi-agent systems are another interesting extension of POMDPs. Examples of such systems are games like poker and Stratego, which involve both imperfect information and game-theoretic reasoning. Finally, one might consider extending work on inverse reinforcement learning to the POMDP setting. In other words, given the dynamics of a POMDP and observing the actions of the agent, we would like to be able to infer the reward function, perhaps for the purpose of user modeling.

POMDPs seem like very difficult computational problems indeed, and yet (perhaps paradoxically) people are continually making good decisions under uncertainty in the real world. Philosophically, if we assume that human intelligence is computational in nature, we must ask ourselves: what accounts for the enormous mismatch between human and computer abilities? It would seem that people are not finding exact solutions to POMDPs in their heads, but are instead exploiting a host of approximation techniques, a rich hierarchical model of the world, flexible multi-resolution memory, and good ability to generalize from experience. The field of artificial intelligence is a long way from creating useful autonomous cognitive agents, but nevertheless, we should draw inspiration from the best autonomous cognitive agents currently known-ourselves.

References

[1] L. Baird and A. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*, pages 968–974, 1998.

[2] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 33–42, 1998.

[3] A. Cassandra. *Exact and approximate algorithms for partially observable Markov decision processes*. PhD thesis, Brown University, Department of Computer Science, 1998.

[4] L. Chrisman. Reinforcement learning with perceptual aliasing: the perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188, 1992.

[5] T. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 118–126, 1998.

[6] M. Dorigo and H. Bersini. A comparison of Q-learning and classifier systems. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 248–255, 1994.

[7] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[8] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[9] N. Hernandez-Gardiol and S. Mahadevan. Hierarchical memory-based reinforcement learning. In *Advances in Neural Information Processing Systems 13*, pages 1047–1053, 2000.

[10] T. Jaakkola, S. Singh, and M. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems 7*, pages 345–352, 1994.

[11] L. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 167–173, 1993.

[12] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

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

[14] M. Kearns, Y. Mansour, and A. Y. Ng. Approximate planning in large pomdps via reusable trajectories. In *Advances in Neural Information Processing Systems 12*, pages 1001–1007, 1999.

[15] M. Kearns, Y. Mansour, and A.Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1324–1331, 1999.

[16] L-J. Lin and T. Mitchell. Memory approaches to reinforcement learning in non-Markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, 1992.

[17] M. Littman. Memoryless policies: theoretical limitations and practical results. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 238–245, 1994.

[18] J. Loch and S. Singh. Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 321–331, 1998.

[19] R. A. McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 190–196, 1993.

[20] R. A. McCallum. Instance-based state identification for reinforcement learning. In *Advances in Neural Information Processing Systems 7*, pages 377–384, 1995.

[21] R. A. McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 387–395, 1995.

[22] N. Meuleau, L. Peshkin, K-E. Kim, and L. Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 427–436, 1999.

[23] A. Moore and C. Atkenson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[24] D. Moriarty, A. Schultz, and J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:199–229, 1999.

[25] R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1088–1094, 1995.

[26] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*, pages 1043–1049, 1998.

[27] L. Peshkin, N. Meuleau, and L. Kaelbling. Learning policies with external memory. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 307–314, 1999.

[28] J. Schmidhuber. Reinforcement learning in Markovian and non-Markovian environments. In *Advances in Neural Information Processing Systems 3*, pages 500–506, 1991.

[29] J. Schmidhuber, J. Zhao, and N. Schraudolph. Reinforcement learning with self-modifying policies. In S. Thrun and L. Pratt, editors, *Learning to learn*, pages 293–309. Kluwer, 1997.

[30] S. Singh, T. Jaakkola, and M. Jordan. Learning without state-estimation in partially observable Markov decision processes. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 284–292, 1994.

[31] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Boston, 1998.

[32] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063, 1999.

[33] A. Teller. Turing completeness in the language of genetic programming with indexed memory. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 136–146, 1994.

[34] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989.

[35] S. Whitehead and D. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

[36] M. Wiering and J. Schmidhuber. HQ-Learning. *Adaptive Behavior*, 6(2):219–246, 1998.

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