Learning Policies with External Memory

Leonid Peshkin
Computer Science Dept.
Brown University, Box 1910
Providence, RI, 02912-1910
ldp@cs.brown.edu

Nicolas Meuleau
Computer Science Dept.
Brown University
nm@cs.brown.edu

Leslie Pack Kaelbling
Computer Science Dept.
Brown University
lpk@cs.brown.edu

Abstract

In order for an agent to perform well in partially observable domains, it is usually necessary for actions to depend on the history of observations. In this paper, we explore a *stigmergic* approach, in which the agent's actions include the ability to set and clear bits in an external memory, and the external memory is included as part of the input to the agent. In this case, we need to learn a reactive policy in a highly non-Markovian domain. We explore two algorithms: SARSA(λ), which has had empirical success in partially observable domains, and VAPS, a new algorithm due to Baird and Moore, with convergence guarantees in partially observable domains. We compare the performance of these two algorithms on benchmark problems.

1 Introduction

A reinforcement-learning agent must learn a mapping from a stream of observations of the world to a stream of actions. In completely observable domains, it is sufficient to look only at the last observation, so the agent can learn a "memoryless" mapping from observations to actions [16]. In general, however, the agent's actions may have to depend on the history of previous observations.

Previous Work There have been many approaches to learning to behave in partially observable domains. They fall roughly into three classes: optimal memoryless, finite memory, and model-based.

The first strategy is to search for the best possible memoryless policy. In many partially observable domains, memoryless policies can actually perform fairly well. Basic reinforcement-learning techniques, such as Q-learning [25], often perform poorly in partially observable domains, due to a very strong Markov assumption. Littman showed [10] that finding the optimal memoryless policy is NP-Hard. However, Loch and Singh [11] effectively demonstrated that techniques, such as SARSA(λ), that are more oriented toward optimizing total reward, rather than Bellman residual, often perform very well. In addition, Jaakkola, Jordan, and Singh [7] have developed an algorithm for finding stochastic memoryless policies, which can perform significantly better than deterministic ones [20].

One class of finite memory methods are the finite-horizon memory methods, which can choose actions based on a finite window of previous observations. For many problems this can be quite effective [12, 18]. More generally, we may use a finite-size memory, which can possibly be infinite-horizon (the systems remembers only a finite number of events, but these events can be arbitrarily far in the past). Wiering and Schmidhuber [26] proposed such an approach, learning a policy that is a finite sequence of memoryless policies.

Another class of approaches assumes complete knowledge of the underlying process, modeled as a *partially observable Markov decision process* (POMDP). Given a model, it is possible to attempt optimal solution [8], or to search for approximations in a variety of ways [5, 4, 6, 13]. These methods can, in principle, be coupled with techniques, such as variations of the Baum-Welch algorithm [17], for learning the model to yield model-based reinforcement-learning systems.

Stigmergy In this paper, we pursue an approach based on *stigmergy*. The term is defined in the Ox-

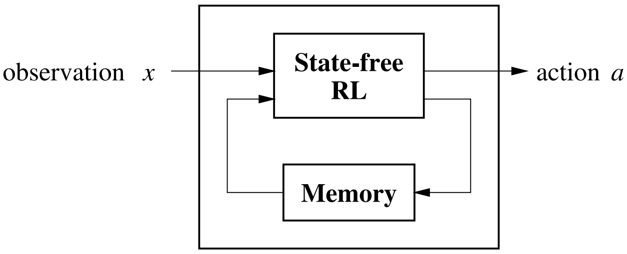

Figure 1: The architecture of a stigmergic policy.

ford English Dictionary [19] as "The process by which the results of an insect's activity act as a stimulus to further activity," and is used in the mobile robotics literature [2] to describe activity in which an agent's changes to the world affect its future behavior, usually in a useful way.

One form of stigmergy is the use of external memory devices. We are all familiar with practices such as making grocery lists, tying a string around a finger, or putting a book by the door at home so you will remember to take it to work. In each case, an agent needs to remember something about the past and does so by modifying its external perceptions in such a way that a memoryless policy will perform well.

We can apply this approach to the general problem of learning to behave in partially observable environments. Figure 1 shows the architectural idea. We think of the agent as having two components: one is a set of memory bits; the other is a reinforcement-learning agent. The reinforcement-learning agent has as input the observation that comes from the environment, augmented by the memory bits. Its output consists of the original actions in the environment, augmented by actions that change the state of the memory. If there are sufficient memory bits, then the optimal memoryless policy for the internal agent will cause the entire agent to behave optimally in its partially observable domain.

Consider, for instance, the load-unload problem represented in Figure 2. In this problem, the agent is a cart that must drive from an *Unload* location to a *load* location, and then back to *unload.* This problem is a simple MDP with a one-bit hidden variable that makes it non-Markov (the agent cannot see whether it is loaded or not). It can be solved using a one-bit external memory: we set the bit when we make the *Unload* observation, and we go right as long as it is set to this value and we do not make the *Load* observation. When we do make the *Load* observation, we clear the

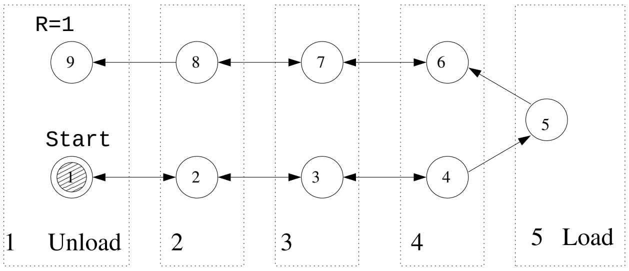

Figure 2: The state-transition diagram of the load-unload problem; aliased states are grouped by dashed boxes.

bit and we go left as long as it stays at value 0, until we reach state 9, getting a reward.

There are two alternatives for designing an architecture with external memory:

- Either we *augment* the action space with actions that change the content of one of the memory bits (adds L new actions if there are L memory bits); changing the state of the memory may require multiple steps.

- Or we *compose* the action space with the set of all possible values for the memory (the size of the action space is then multiplied by 2^L, if there are L bits of memory). In this case, changing the external memory is an instantaneous action that can be done at each time step in parallel with a primitive action, and hence we can reproduce the optimal policy of the load-unload problem, without taking additional steps.

Complexity considerations usually lead us to take the first option. It introduces a bias, since we have to lose at least one time-step each time we want to change the content of the memory. However, it can be fixed in most algorithms by not discounting memory-setting actions.

The external-memory architecture has been pursued in the context of classifier systems [3] and in the context of reinforcement learning by Littman [10] and by Martín [15]. Littman's work was model-based; it assumed that the model was completely known and did a branch-and-bound search in policy space. Martín worked in the model-free reinforcement-learning domain; his algorithms were very successful at finding good policies for very complex domains, including some simulated visual search and block-stacking tasks. However, he made a number of strong assumptions and restrictions: task domains are strictly goal-oriented; it is assumed that there is a deterministic policy that achieves the goal within some specified number of steps from every initial state; and there is no desire for optimality in path length.

This work We were inspired by the success of Martín's algorithm on a set of difficult problems, but concerned about its restrictions and a number of details of the algorithm that seemed relatively *ad hoc*. At the same time, Baird and Moore's work on VAPS [1], a general method for gradient descent in reinforcement learning, appealed to us on theoretical grounds. This paper is the result of attempting to apply VAPS algorithms to stigmergic policies, and understanding how it relates to Martín's algorithm. In this process, we have derived a much simpler version of VAPS for the case of highly non-Markovian domains: we calculate the same gradient as VAPS, but with much less computational effort.

In the next section, we present the relevant learning algorithms. Then we describe a set of experimental domains and discuss the relative performance of the algorithms.

2 Algorithms

We begin by describing the most familiar of the algorithms, SARSA(λ). We then describe the VAPS algorithm in some detail, followed by our simplified version.

2.1 SARSA(λ)

SARSA is an on-policy temporal-difference control learning algorithm [23]. Given an experience in the world, characterized by starting state x, action a, reward r, resulting state x' and next action a', the update rule for SARSA(0) is

$$Q(x,a) \leftarrow Q(x,a) + \alpha[r + \gamma Q(x',a') - Q(x,a)] \ . \quad (1)$$

It differs from the classical Q-learning algorithm [24] in that, rather than using the maximum Q-value from the resulting state as an estimate of that state's value, it uses the Q-value of the resulting state and the action that was actually chosen in that state. Thus, the values learned are sensitive to the policy being executed.

In truly Markov domains, Q-learning is usually the algorithm of choice; policy-sensitivity is often seen as a liability, because it makes issues of exploration more complicated. However, in non-Markov domains, policy-sensitivity is actually an asset. Because observations do not uniquely correspond to underlying states, the value of a policy depends on the distribution of underlying states given a particular observation. But this distribution generally depends on the policy. So, the value of a state, given a policy, can only be evaluated *while executing that policy*. In fact, Q-learning can be shown to fail to converge on very simple non-Markov domains [22]. Note that, when SARSA is used in a non-Markovian environment, the symbols x and x' in equation (1) represent observations, which usually can correspond to several states.

The SARSA algorithm can be augmented with an eligibility trace, to yield the SARSA(λ) algorithm (a detailed exposition is given by Sutton and Barto [23].) With the parameter λ set to 0, SARSA(λ) is just SARSA. With λ set to 1, it is a pure Monte Carlo method, in which, at the end of every trial, each state-action pair is adjusted toward the cumulative reward received on this trial after the state-action pair occurred. Pure Monte-Carlo algorithms make no attempt at satisfying Bellman equations relating the values of subsequent states; in partially observable domains, it is often impossible to satisfy the Bellman equation, making Monte-Carlo a reasonable choice. SARSA(λ) describes a useful class of algorithms, then, with appropriate choice of λ depending on the problem. Thus, SARSA(λ) with a large value of λ seems like the most appropriate of the conventional reinforcement-learning algorithms for solving partially-observable problems.

2.2 VAPS

Baird and Moore have derived, from first principles, a class of stochastic gradient-descent algorithms for reinforcement learning. At the most abstract level, we seek to minimize some measure of the expected cost of our policy; we can describe this high-level criterion as

$$B = \sum_{T=0}^{\infty} \sum_{\tilde{s} \in \tilde{S}_T} \Pr(\tilde{s}) \varepsilon(\tilde{s}) \ ,$$

where $\tilde{S}_T$ is the set of all possible experience sequences that terminate at time T. That is,

$$\tilde{s} = \langle x_0, u_0, r_0, ..., x_t, u_t, r_t, ..., x_T, u_T, r_T \rangle \ ,$$

where x_t, u_t, and r_t are the observation, action, and reward at step t of the sequence, and x_T is an observation associated with a terminal state. The loss incurred by a sequence $\tilde{s}$ is $\varepsilon(\tilde{s})$. We restrict our attention to time-separable loss functions, which can be written as

$$\varepsilon(\tilde{s}) = \sum_{t=0}^{T} e(\mathrm{trunc}(\tilde{s}, t)) \, , \qquad \text{for all } \tilde{s} \in \tilde{S}_T \, ,$$

where $e(s)$ is an instantaneous error function associated with each (finite) sequence prefix $s = \langle x_0, u_0, r_0, ..., x_t, u_t, r_t \rangle$ (x_t being any observation, not necessarily a terminal one), and $\mathrm{trunc}(\tilde{s}, t)$ representing the sequence $\tilde{s}$ truncated after time t. For instance, an error measure closely related to Q-learning is the squared Bellman residual:

$$\begin{aligned} e_{\mathrm{QL}}(s) = & \frac{1}{2} \sum_x \Pr(x_t = x \mid x_{t-1}, u_{t-1}) \\ & [r_{t-1} + \max_u \gamma Q(x, u) - Q(x_{t-1}, u_{t-1})]^2 \, . \end{aligned}$$

The SARSA version of the algorithm uses the following error measure:

$$\begin{aligned} & e_{\mathrm{SARSA}}(s) = \\ & \quad \frac{1}{2} \sum_x \Pr(x_t = x \mid x_{t-1}, u_{t-1}) \sum_u \Pr(u_t = u \mid x_t) \\ & \quad [r_{t-1} + \gamma Q(x, u) - Q(x_{t-1}, u_{t-1})]^2 \, . \end{aligned}$$

Note that we average over all possible actions u_t according to their probability of being chosen by the policy instead of picking the one that maximizes Q-values as in e_{QL}. Baird and Moore also consider a kind of policy search, which is analogous to REINFORCE [27]:

$$e_{\mathrm{policy}}(s) = b - \gamma^t r_t \, ,$$

where b is any constant. This immediate error is summed over all time t, leading to a summation of all discounted immediate rewards $\gamma^t r_t$. In order to obtain the good properties of both criteria, they construct a final criterion that is a linear combination of the previous two:

$$e = (1 - \beta) e_{\mathrm{SARSA}} + \beta e_{\mathrm{policy}} \, .$$

This criterion combines Value And Policy Search and is, hence, called VAPS. We will refer to it as VAPS(β), for different values of β.

Baird and Moore show that the gradient of the global error with respect to weight k can be written as:

$$\begin{aligned} & \frac{\partial}{\partial w_k} B = \sum_{t=0}^{\infty} \sum_{s \in S_t} \Pr(s) \\ & \quad \left[\frac{\partial}{\partial w_k} e(s) + e(s) \sum_{j=1}^{t} \frac{\partial}{\partial w_k} \ln \Pr(u_{j-1} | x_{j-1}) \right] \, , \end{aligned}$$

where S_t is the set of all experience prefixes of length t. Technically, it is necessary that $\Pr(u^t = u \mid x_t = x) > 0$ for all (x, u) (otherwise, some zero probability trajectories may have a non-zero contribution to the gradient of B [9]). In this work we use the Boltzmann law for picking actions, which guarantees this property (see section 2.3).

It is possible to perform stochastic gradient descent of the "error" B, by repeating several trials of interaction with the process. Each experimental trial of length T provides one sample of $s \in S_t$ for each $t \leq T$. Of course, these samples are not independent, but it does not matter since we are summing them and not multiplying them. We are thus using stochastic approximation to estimate the expectation over $s \in S_t$ in the above equation. During each trial, the weights are kept constant and the approximate gradients of the error at each time t,

$$\frac{\partial}{\partial w_k} e(s) + e(s) \sum_{j=1}^{t} \frac{\partial}{\partial w_k} \ln \Pr(u_{j-1} | x_{j-1}) \, ,$$

are accumulated. Weights are updated at the end of each trial, using the sum of these immediate gradients. An incremental implementation of the algorithm can be obtained by using, at every step t, the following update rules:

$$\begin{aligned} \Delta T_{k,t} &= \frac{\partial}{\partial w_k} \ln \Pr(u_{t-1} | x_{t-1}) \, , \\ \Delta w_k &= -\alpha \left[\frac{\partial}{\partial w_k} e(\tilde{s}_t) + e(\tilde{s}_t) T_{k,t} \right] \, , \end{aligned}$$

where s_t represents the experience prefix $\langle x_0, u_0, r_0, \ldots, x_t, u_t, r_t \rangle$, i.e., the history at time t. Note that the "exploration trace" $T_{k,t}$ is independent of the immediate error e used. It only depends on the way the output $\Pr(u_t = u | x_t)$ varies with the weights w_k, i.e., on the representation chosen for the policy.

The gradient of the immediate error e with respect to the weight w_k is easy to calculate. For instance, in the case of the SARSA variant of the algorithm we have:

$$\begin{aligned} & \frac{\partial}{\partial w_k} e_{\mathrm{SARSA}}(s) = \\ & \quad \sum_x \Pr(x_t = x \mid x_{t-1}, u_{t-1}) \sum_u \Pr(u_t = u \mid x_t) \\ & \quad [r_{t-1} + \gamma Q(x, u) - Q(x_{t-1}, u_{t-1})] \\ & \quad \left[\gamma \frac{\partial}{\partial w_k} Q(x, u) - \frac{\partial}{\partial w_k} Q(x_{t-1}, u_{t-1}) \right] \, . \end{aligned}$$

Once more, we descend this gradient by stochastic approximation: the averaging over x_t and u_t is replaced by a sampling of these quantities. However, since these variables appear twice in the equation and they are not just added, we have to sample both x_t and u_t independently two times in order to avoid any bias in the estimation of the gradient. It is not realistic to satisfy this requirement in a truly on-line situation, since the only way to get a new observation is by actually *performing* the action. Note that for the case $\beta = 1$ we do not need the second sample, so the VAPS(1) algorithm is effective in the on-line case.

In the case of policy search we have: $\frac{\partial}{\partial w_k} e_{\text{policy}}(s) = 0$, for all w_k. This may seem strange; but for policy search, the important thing is the state occupations, which enter into the weight updates through the trace.

2.3 VAPS(1)

In this section, we explore a special case of VAPS, in which the Q-values are stored in a look-up table. That is, there is one weight $w_k = Q(x,u)$ for each state-action pair. Note that it is not necessary to use the VAPS sequence-based gradient in a look-up table implementation of QL or SARSA, as long as it is confined to a Markovian environment. However, it makes sense to use it in the context of POMDPs. Under this hypothesis, the exploration trace $T_{k,t}$ associated with each parameter $Q(x,u)$ will be written $T_{x,u,t}$.

We will also focus on a very popular rule for randomly selecting actions as a function of their Q-value, namely the Boltzmann law:

$$Pr(u_t = u | x_t = x) = \frac{e^{Q(x,u)/c}}{\sum_{u'} e^{Q(x,u')/c}} ,$$

where c is a temperature parameter.[1] Under this rule we get:

$$\frac{\partial \ln \Pr(u_t = u | x_t = x)}{\partial Q(x', u')} =$$

$$\begin{cases} 0 & \text{if } x' \neq x, \\ -\Pr(u_t = u' | x_t = x)/c & \text{if } x' = x \text{ and } u' \neq u, \\ \left[1 - \Pr(u_t = u | x_t = x)\right]/c & \text{if } x' = x \text{ and } u' = u. \end{cases}$$

In this case, and if we add the hypothesis that the problem is an achievement task, i.e., the reward is always 0 except when we reach an absorbing goal state, the exploration trace $T_{x,u,t}$ takes a very simple form:

$$\begin{aligned} T_{x,u,t} &= \tfrac{1}{c}\left[N^t_{x,u} - N^t_x \Pr(u_t = u | x_t = x)\right] \\ &= \tfrac{1}{c}\left[N^t_{x,u} - E[N^t_{x,u}]\right] , \end{aligned} \quad (2)$$

where $N^t_{x,u}$ is the number of times that action u has been executed in state x at time t, N^t_x is the number of times that state x has been visited at time t, and $E[N^t_{x,u}]$ represents the expected number of times we should have performed action u in state x, knowing our exploration policy and our previous history.

As a result of equation (2), VAPS using e_{policy} as immediate error, look-up tables and Boltzmann exploration reduces to a very simple algorithm. At each time-step where the current trial does not complete, we just increment the counter $N^t_{x,u}$ of the current state-action pair. When the trial completes, this trace is used to update all the Q-values, as described above.

It is interesting to try to understand the properties and implications of this simple rule. First, a direct consequence is that when something surprising happens, the algorithm adjusts the unlikely actions more than the likely ones. In other words, this simple procedure is very intuitive, since it assigns credit to state-action pairs proportional to the deviation from the expected behaviour. Note that SARSA(λ) is not capable of such a discrimination. This difference in behaviour is illustrated in the simulation results.

A second interesting property is that the Q-value updates tend to 0 as the length of the trial tends to infinity. This also makes sense, since the longer the trial, the less the final information received (the final reward) is relevant in evaluating each particular action. Alternatively, we could say that when too many actions have been performed, there is no reason to attribute the final result more to one of them than to others. Finally, unlike with Baird and Moore's version of the Boltzmann law, the sum of the updates to the Q-values on every step is 0. This makes it more likely that the weights will stay bounded.

3 Experiments

Domains We have experimented with SARSA and VAPS on five simple problems. Two are illustrative problems previously used in the reinforcement-learning literature; two others are instances of load-unload with different parameters; and the fifth is a variant of load-unload designed by us in an attempt to demonstrate a situation in which VAPS might outperform SARSA. The five problems are :

[1] Note that Baird and Moore use an unusual version of the Boltzmann law, with $1+e^x$ in place of e^x in both the numerator and the denominator. We have found that it complicates the mathematics and worsens the performance, so we will use the standard Boltzmann law throughout.

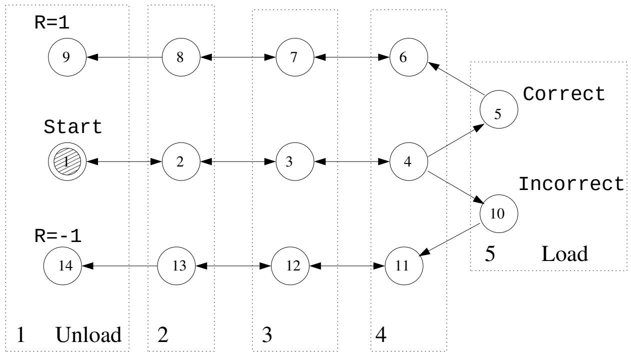

Figure 3: The state-transition diagram of the load-unload problem with two loading locations; aliased states are grouped by dashed boxes.

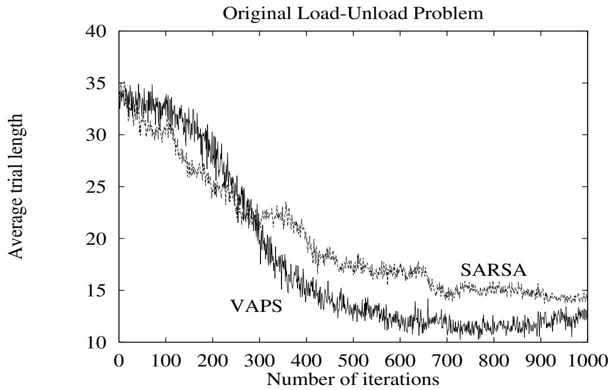

Figure 4: Learning curves for VAPS and SARSA on the load-unload problem (one loading location).

- Baird and Moore's problem [1], designed to illustrate the behavior of VAPS,
- McCallum's 11-state maze [12], which has only 6 observations.
- The load-unload problem, as described above, in which there are three locations (the loading location, the unloading location, and one intermediate one),
- A five-location load-unload problem (fig. 2), and
- A variant of the load-unload problem where a second loading location has been added, and the agent is punished instead of rewarded if it gets loaded at the wrong location. The state space is shown in figure 3; states contained in a box are observationally indistinguishable to the agent. The idea here is that there is a single action that, if chosen, ruins the agent's long-term prospects. If this action is chosen due to exploration, then SARSA(λ) will punish all of the action choices along the chain but VAPS will punish only that action.

All these domains have a single starting state, except McCallum's problem, where the starting state is chosen uniformly at random.

Algorithmic Details For each problem, we ran two algorithms: VAPS(1) and SARSA(1). The optimal policy for Baird's problem is memoryless, so the algorithms were applied directly in that case. For the other problems, we augmented the input space with an additional memory bit, and added two actions: one for setting the bit and one for clearing it.

The Q-functions were represented in a table, with one weight for each observation-action pair. The learning rate is determined by a parameter, α_0; the actual learning rate has an added factor that decays to 0 over time: $\alpha = \alpha_0 + \frac{1}{10N}$, where N is trial number. The temperature was also decayed in an *ad hoc* way, from $c_{\max}$ down to $c_{\min}$ with an increment of

$$\delta c = \left(\frac{c_{min}}{c_{max}}\right)^{1/(N-1)}$$

on each trial. In order to guarantee convergence of SARSA in MDPs, it is necessary to decay the temperature in a way that is dependent on the Q-values themselves [21]; in the POMDP setting it is much less clear what the correct decay strategy is. In any case, we have found that the empirical performance of the algorithm is not particularly sensitive to the temperature. The parameter b in the immidiate error e_{policy} of VAPS was always set to 0.

Experimental Protocol Each learning algorithm was executed for K runs; each run consisted of N trials, which began at the start state and executed until a terminal state was reached or M steps were taken. If the run was terminated at M steps, it was given a terminal reward of -1; M was chosen, in each case, to be 4 times the length of the optimal solution. At the beginning of each run, the weights were randomly reinitialized to small values.

Results It was easy to make both algorithms work well on the first three problems: Baird's, McCallum's and small load-unload. The algorithms typically converged in fewer than 100 runs to an optimal policy. One thing to note here is that our version of VAPS, using the true Boltzmann exploration distribution rather

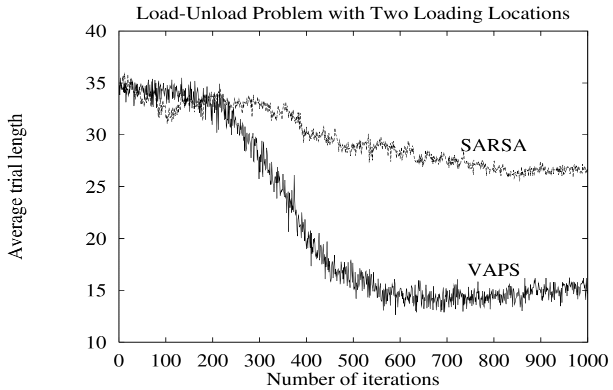

Figure 5: Learning curves for VAPS and SARSA on the load-unload problem with two loading locations.

than the one described by Baird and Moore, seems to perform significantly better than the original, according to results in their paper.

Things were somewhat more complex with the last two problems (5 location load-unload with one or two loading locations). We experimented with parameters over a broad range and determined the following:

- VAPS requires a value of β equal or very nearly equal to 1; these problems are highly non-Markovian, so the Bellman error is not at all useful as a criterion.
- For similar reasons, $\lambda = 1$ is best for SARSA(λ).
- Exploration was simplified by setting ϵ to 0; empirically, $c_{max} = 1.0$ and $c_{min} = 0.2$ worked well for VAPS in both problems, and $c_{max} = 0.2$, $c_{min} = 0.1$ worked well for SARSA(λ).
- A base learning rate of $\alpha_0 = 0.5$ worked well for both algorithms in both domains.

Figures 4 and 5 show learning curves for both algorithms, averaged over 50 runs, on the load-unload problem with one or two loading locations. Each run consisted of 1,000 trials. The vertical axis shows the number of steps required to reach the goal, with the terminated trials considered to have taken M steps.

On the original load-unload problem, the algorithms perform essentially equivalently. Most runs of the algorithm converge to the optimal trial length of 9 and stay there; occasionally, however it reaches 9 and then diverges. This can probably be avoided by decreasing the learning rate more steeply. When we add the second loading location, however, there is a significant difference. VAPS(1) consistently converges to a near-optimal policy, but SARSA(1) does not. The idea is that sometimes, even when the policy is pretty good, the agent is going to pick up the wrong load due to exploration and get punished for it. SARSA will punish all the state-action pairs equally; VAPS(1) will punish the bad state-action pair more due to the different principle of credit assignment.

4 Conclusions

As Martín and Littman showed, small POMDPs can be solved effectively using stigmergic policies. Learning reactive policies in highly non-Markovian domains is not yet well-understood. We have seen that the VAPS algorithm, somewhat modified, can solve a collection of small POMDPs, and that although SARSA(λ) performs well on some POMDPs, it is possible to construct cases on which it fails. In a generalization of this work, we applied the VAPS algorithm to the problem of learning general finite-state controllers (which encompass external-memory policies) for POMDPs [14].

Acknowledgments

This work was supported in part by DARPA/Rome Labs Planning Initiative grant F30602-95-1-0020.

References

[1] Leemon C. Baird and Andrew W. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*. The MIT Press, 1999.

[2] R. Beckers, Owen E. Holland, and J. L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In *Artificial Life IV*, Cambridge, Massachusetts, 1994. The MIT Press.

[3] L.B. Booker, D.E. Goldberg, and J.H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40, 1989.

[4] E.A. Hansen. *Finite-Memory Control of Partially Observable Systems*. PhD thesis, Department of Computer Science, University of Massachusetts at Amherst, 1998.

[5] E.A. Hansen. Solving POMDPs by searching in policy space. In *Proceedings of the Eighth Con-*

ference on Uncertainty in Artificial Intelligence, pages 211–219, Madison, WI, 1998.

[6] Milos Hauskrecht. *Planning and Control in Stochastic Domains with Imperfect Information.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1998.

[7] T. Jaakkola, S. Singh, and M.R. Jordan. Reinforcement learning algorithm for partially observable Markov problems. In *Advances in Neural Information Processing Systems, 7.* The MIT Press, Cambridge, MA, 1994.

[8] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

[9] L.P. Kaelbling, K.E. Kim, N. Meuleau, and L. Peshkin. Searching for finite-state POMDP controllers. Technical Report CS-99-06, Brown University, 1999.

[10] Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In *From Animals to Animats 3*, Brighton, UK, 1994.

[11] John Loch and Satinder 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*, Madison, Wisconsin, 1998.

[12] Andrew Kachites McCallum. *Reinforcement Learning with Selective Perception and Hidden State.* PhD thesis, University of Rochester, Rochester, New York, 1995.

[13] N. Meuleau, K.E. Kim, A.R. Cassandra, and L.P. Kaelbling. Solving POMDPs by searching the space of finite policies. *Fifteenth Conference on Uncertainty in Artificial Intelligence*, Submitted, 1999.

[14] N. Meuleau, L. Peshkin, K.E. Kim, and L.P. Kaelbling. Learning finite-state controllers for partially observable environments. *Fifteenth Conference on Uncertainty in Artificial Intelligence*, Submitted, 1999.

[15] Mario Martín. *Reinforcement Learning for Embedded Agents facing Complex Tasks.* PhD thesis, Universitat Politecnica de Catalunya, Barcelona, Spain, 1998.

[16] Martin L. Puterman. *Markov Decision Processes.* John Wiley & Sons, New York, 1994.

[17] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[18] Mark B. Ring. *Continual Learning in Reinforcement Environments.* PhD thesis, University of Texas at Austin, Austin, Texas, August 1994.

[19] John Simpson and Edmund Weiner, editors. *Oxford English Dictionary.* Oxford University Press, second edition, 1989.

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

[21] Satinder Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvari. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, to appear.

[22] Satinder Pal Singh, Tommi Jaakkola, and Michael I. Jordan. Model-free reinforcement learning for non-Markovian decision problems. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 284–292, San Francisco, California, 1994. Morgan Kaufmann.

[23] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, Massachusetts, 1998.

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

[25] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[26] Marco Wiering and Juergen Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2), 1997.

[27] Ronald J. Williams. A class of gradient-estimating algorithms for reinforcement learning in neural networks. In *Proceedings of the IEEE First International Conference on Neural Networks*, San Diego, California, 1987.