Credit Assignment Method for Learning Effective Stochastic Policies in Uncertain Domains

Sachiyo Arai and Katia Sycara
The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213 USA
E-Mail: {sachiyo, katia}@cs.cmu.edu

Abstract

In this paper, we introduce *FirstVisit Profit-Sharing* (FVPS) as a credit assignment procedure, an important issue in classifier systems and reinforcement learning frameworks. FVPS reinforces effective rules to make an agent acquire stochastic policies that cause it to behave very robustly within uncertain domains, without pre-defined knowledge or subgoals. We use an internal episodic memory, not only to identify perceptual aliasing states but also to discard looping behavior and to acquire effective stochastic policies to escape perceptual deceptive states.

We demonstrate the effectiveness of our method in some typical classes of Partially Observable Markov Decision Processes, comparing with Sarsa(λ) using a replacing eligibility trace. We claim that this approach results in an effective stochastic or deterministic policy which is appropriate for the environment.

1 Introduction

In this paper, we present the learning algorithm to address the *credit assignment procedure*, which is a very important issue in both classifier systems and reinforcement learning frameworks [Holland, 1986][Moriarty et al., 1999]. We focus on the environments that have goals to be attained by autonomous agents that learn with delayed reward. This problem can be called a sequential decision problem, which is defined by a set of an agent's sensory observations and a set of actions that map observations to successor observations. If an agent's sensory observations provide the complete state of its environment, the environment can be formulated as Markov decision processes (MDPs), for which a number of very successful planning[Barto et al., 1995] and reinforcement learning[Watkins & Dayan, 1992] approaches have been developed.

However, in many real environments, such as multi-agent and distributed control mobile robotics' environments, only partial information about the state-spaces can be expected. These environments can be formulated as partially observable Markov decision processes (POMDPs) where agents suffer from hidden states or perceptual aliasing (i.e., the agent takes some different environmental states as the same sensory observation). Therefore, finding an efficient method for solving POMDPs would be a very practical contribution to creating adaptive agents.

Recent approaches in POMDPs can be classified into two types. One is called a memory-based approach[Chrisman, 1992][McCallum, 1995], which attempts to overcome perceptual aliasing by using memory to estimate real-state and finally to find deterministic policies for the environment. The other is called a memory-less approach[Jaakkola et al., 1994] [Loch & Singh, 1998][Singh et al., 1994], which can acquire a stochastic policy to make the agent robust against perceptual aliasing. Recently, there also is an intermediate approach[Lanzi, 2000] which introduces small internal memories, not to construct a model of the environment but to retain effective classifiers by combining these memories with a genetic algorithm.

Sarsa(λ) using a replacing eligibility trace, proposed in [Singh & Sutton, 1996], is highly regarded in [Peshkin et al., 1999][Lanzi, 2000][Loch & Singh, 1998] as a memory-less approach which can work very well in POMDPs. Unfortunately, their results refer to different testbeds, so we cannot see what the resolvable sub-class of POMDPs is by this method.

In this paper, we abstract three interesting subclasses

of POMDPs which bring agents serious confusions and propose a *FirstVisit Profit-Sharing* (FVPS) approach to make agents behave robustly in these confusions. We show the performance of our approach by comparison with Sarsa(λ) in an *episodic task* where there is a goal within a finite number of steps from every initial state. The episodic task is common in the real world, where we can define a desirable situation as a goal but cannot define the subgoals.

FVPS is classified as a memory-less approach. Although we use an internal one-dimensional episodic memory, this memory is not used to construct the state-transition model but only to discard looping behavior and to acquire an effective stochastic policy to escape perceptual deceptive states. Therefore, FVPS does not cost much computation and memory space. It is very similar to the Monte-Carlo approach and Sarsa(1) in that it makes no attempt at satisfying Bellman equations relating the values of successive states. It is different from Monte-Carlo in that the weight[1] of rules acquired by our method has no meaning as the estimation of state-action values, whereas Monte-Carlo attempts to estimate the value of the state (or state-action pair in Sarsa(λ)) as an averaged reward. These properties of FVPS not only make an agent behave robustly against perceptual aliasing but also save memory and computation costs by finding and retaining only rules essential for surviving in the environment.

In Section 2, we describe our domain, notations, and related algorithms. Section 3 introduces our approach, FVPS. An empirical comparison of performance using two learning approaches, FVPS and Sarsa(λ=0, 0.9, 1), is presented via several experiments in Section 4, and we discuss the applicability and effectiveness of our approach for the real world in Section 5. Finally, we conclude and summarize our future work.

2 Problem Domain

2.1 Agent Model

The agent is modeled as a reinforcement learning entity engaged in an episodic task in an unknown environment, where there are no intermediate subgoals for which intermediate rewards can be given. (Note: Because our focus here is on *credit assignment*, the agent does not have any genetic algorithm framework.) An environment is defined by a finite set of state S, the agent has a finite set of actions A, and the agent's sensors provide its observations from a finite set X. Each agent consists of five modules: a *State Recognizer*, a *LookUp Table*, an *Action Selector*, an *Episodic Memory* and the *Learner*, which includes the credit assignment procedure, as shown in Fig.1.

[1]Weight here is similar in meaning to *Value* in the DP-based approaches, and *Strength* in the classifier systems.

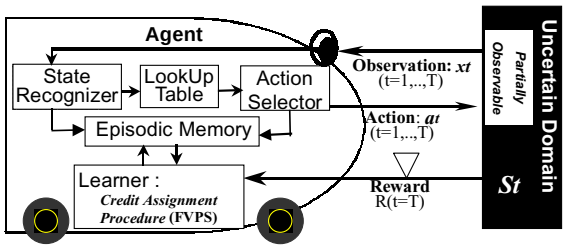


Figure 1: Agent Model

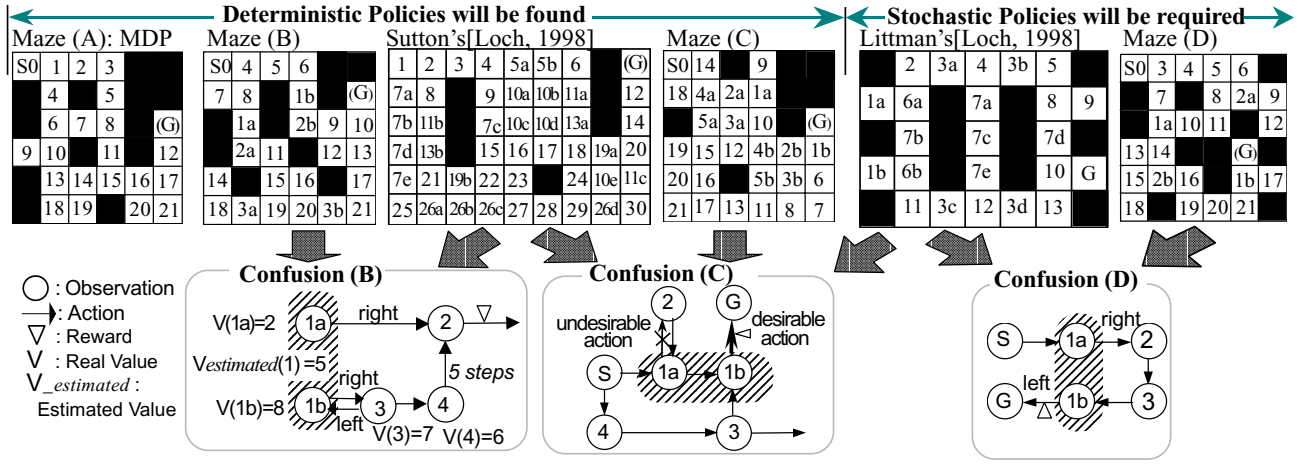
Initially, the agent observes x_t as s_t, the partially available state of its environment at time t. An action is then selected (using a certain exploration method) from the action set A_t, which contains all the available actions at time t. If there is no reward after action a_t, the agent stacks the observation-action pair, (x_t, a_t), into its *Episodic Memory*, and repeats this cycle until a reward is generated. The observation-action pair is called a *rule* in this paper. The process of moving from a start state to the final reward state is called an *episode*.

2.2 Target classes of POMDPs

We take five mazes, including two test problems which are treated in [Loch & Singh, 1998], as shown in Fig.2, to show the typical confusions which cause improper behavior of the agent. Besides the mazes we treat here, the *load-unload problem*[Peshkin et al., 1999] and *Woods101, Maze7, Woods101$\frac{1}{2}$, and Woods102* [Lanzi & Wilson, 2000], which includes confusion type (D), have been used in researching POMDPs.

Except for maze (A), they include some perceptual aliasing areas (such as 1a, 1b,..and 26d), which make the agent fall into the confusion types (B), (C), and (D). For mazes (B) and Sutton's, a common action can be effective among the same observations, and for maze (C), there exists another reliable route which does not include the aliasing area, so a stochastic policy is not required. Therefore, in mazes of (B), (C) and Sutton's, a deterministic policy can be found in each observation, regardless of existing confusions.

However, in maze (D), which includes confusion type (D), the agent cannot achieve the goal with only



For Types A-D and Sutton's, the agent can observe its eight neighboring cells and (G) indicates the goal position of each environment, but the agent cannot observe it as a goal position. In Littman's environment, the agent can observe its four directions: up, right,down and left; if G is in one of these four directions, it can observe G as a goal. Maze (A), (B) and (C) have been reproduced from [Miyazaki and Kobayashi 1999].

Figure 2: Target subclasses of POMDPs and their Confusion Types

a deterministic policy. For example, if the agent is in state $1a$, *down* is a desirable action, but if it is in state $1b$, it needs to move *up* to reach the goal. Also, stochastic policies are necessary in Littman's maze[Loch & Singh, 1998], because the observations are noisy, with the agent getting the correct observation only 70% of the time.

2.3 Related Algorithms

There are two types of credit assignment procedures. One is inspired by dynamic programming (e.g., [Watkins & Dayan, 1992]). The other is inspired by Holland's learning classifier systems (e.g., [Grefenstette, 1988]). The former one basically attempts to satisfy Bellman equations relating the values of successive states (or state-action pairs) to make an agent behave optimally. The latter one does not attempt to estimate the value of all rules that cover the state space, but just accumulates the weight on successful rules based on the agent's experiences. These credit assignment procedures are called the *bootstrapped* method and the *non-boot-strapped* method, respectively. The equivalence between *bucket brigade algorithm*, used in the classifier system[Holland, 1986], and Q-learning is proved in [Dorigo & Bersini, 1994], so we can't say that all classifier systems apply the *non-boot strapped* method.

Q-learning by [Watkins & Dayan, 1992](Eq.1) computes by successive approximations a table of all values $Q(x,a)$. At each time step in the episode n the agent updates $Q_n(x_t,a_t)$ by recursively discounting future utilities and weighting them by positive learning rate α. Here $\gamma(0 < \gamma < 1)$ is a discount parameter. If there is no immediate reward r, the agent uses $r = 0$ to update $Q_n(x_t,a_t)$.

Q-learning often performs poorly in POMDPs due to computing by successive approximations. Sarsa($\lambda = 0$) also updates the value of the state using successive state values, as shown in Eq.2. However, [Loch & Singh, 1998] demonstrated that Sarsa using a replacing eligibility trace (as shown in Eq.3) with a large λ value (such as $\lambda > 0.9$) performs well in some classes of POMDPs.

Q-learning:

$$Q_{n+1}(x_t,a_t) \leftarrow (1-\alpha)\cdot Q_n(x_t,a_t) + \alpha(r + \gamma max_{b \in actions} Q_n(x_{t+1},b)) \quad (1)$$

Sarsa(λ):

$$Q_{n+1}(x_t,a_t) \leftarrow (1-\alpha)\cdot Q_n(x_t,a_t) + \alpha(r + \gamma Q_n(x_{t+1},a_{t+1})) \quad (2)$$

Replacing Eligibility Trace and Sarsa(λ):

$$\begin{aligned} &1. \quad \delta t \leftarrow r_t + \gamma Q_n(x_{t+1},a_{t+1}) - Q_n(x_t,at) \\ &2. \quad \eta_t(x_t,a_t) = 1 \\ &3. \quad \forall(x \neq x_t, a \neq a_t); \eta_t(x,a) = \gamma\lambda\eta_{t-1}(x.a) \quad (3) \\ &4. \quad \forall(x,a); Q_{n+1}(x,a) \leftarrow Q_n(x,a) + \alpha\cdot\delta_t\cdot\eta(x,a) \end{aligned}$$

The eligibility traces are initialized to zero, and in episodic tasks they are reinitialized to zero after every episode. When $\lambda = 1$ in Sarsa, it is the same as the Monte-Carlo method[Singh & Sutton, 1996], which does not attempt to satisfy Bellman equations

relating the values of successive states. It seems that using a successive value for state-value estimation is not effective for POMDPs. Here, we should note that Sarsa($\lambda > 0$) requires computation time to update the whole table of experienced rules, or in the most serious case, must update all state spaces at each step.

Profit-Sharing by [Grefenstette, 1988], used in the classifier system, provides a hopeful *non-boot strapped* credit assignment method. Profit-Sharing is very similar to the Monte-Carlo method in that it does not utilize successive approximations to compute a table of all state values. This is an important property which can make Profit-Sharing attractive when one requires the value of only a subset of the state.

3 Our Approach

3.1 Requirements for Acquiring a Proper Policy

There are three requirements in the algorithm to make an agent behave robustly in uncertain domains. The first is that the algorithm needs to update the weight of the state independently without using successive state values. The second is that the algorithm should not attempt to estimate the value function. This will fail in POMDPs in which the agent's sensory input is limited. The value estimation, mapping one value to one rule, makes no sense when there is no unique value to an observation. The accumulation of the weight on successful rules is enough to make policies proper. The third requirement is that the algorithm must guarantee that the agent will reach the goal within a finite number of steps.

Our credit assignment approach is based on Profit-Sharing[Grefenstette, 1988]. The good properties which FVPS inherits from Profit-Sharing satisfy the first and second requirements mentioned above. However, the Profit-Sharing approach does not take the infinite loops in the agent's episode explicitly into consideration because the Profit-Sharing is assumed to use in combination with a genetic algorithm which will get rid of any bad behavior.

FVPS improves on Profit-Sharing in that it guarantees that loops are discarded without evolutionary pressure. In much reinforcement learning research, the target problems do not contain loops (e.g., board games), although there are some problems which do contain loops[Hansen, 1998]. These loops may result in the agent exhibiting improper behavior with respect to achieving its goal. In general, it is important to pursue *proper* policy rather than *optimal* for POMDPs. A *proper* policy is one that is guaranteed to converge on a solution; i.e., the agent should not become trapped within infinite loops in the state machine. To guarantee convergence on a *proper* policy in POMDPs, we introduce the *FirstVisit* routine and credit assignment function.

To illustrate the advantage of this point, consider the example in Fig. 2 Confusion (B). The state value, V, represents the minimum number of steps to a reward. In this example, the highest value of V is 1. The values of states 1a and 1b, V(1a) and V(1b), are 2 and 8, respectively. Although these two states are different, they are perceived by the agent as being the same state (i.e., state 1). If the agent moves to state 1a and 1b with equal weight, $\mathsf{V}(1) = \frac{2+8}{2} = 5$. Therefore the value of state 1 is equal to the value of state 3, i.e., $\mathsf{V}(3) = 5$. If the agent uses these state values according to a DP-based algorithm, such as Q-learning (Eq. 1), it will move *left* into state 3. Otherwise, the agent moves *right* into state 1. This means that the agent learns the improper policy in which it only transits between states 1b and 3. If the agent only reinforces the successful rules without any propagation to other rules, it can escape this looping behavior caused by confusion type (B).

3.2 FirstVisit Profit-Sharing

Our solution to this problem is very simple. If the current observation is the revisited one and the same action is executed, the agent does not stack this rule into the *Episodic Memory*, since the re-executed rule will cause a cyclic loop in the agent's route. This routine does not require any extra memory other than that used by the current framework of the Profit-Sharing algorithm, where an *Episodic Memory* is used to accumulate rules until the goal is achieved.

Fig. 3 shows our algorithm. The *FirstVisit* routine prohibits the agent from reinforcing the weight of the rules which make up the loop, and can retain essential rules for a stochastic policy. Consider the episode illustrated in Fig. 4(1), which is one example of a generated decision sequence in the initial stage of learning, in the maze (C) shown in Fig. 2(C). In this episode, the sub-sequence $(1a, Up) - (9, Down) .. - (9, Down)$ forms an obviously needless loop; therefore the weight of $(1a, Up)$ should not be reinforced, while $(1b, Up)$ is necessary to reach the goal. However, the agent cannot tell the difference between $1a$ and $1b$, so we need to consider which rule should be retained and which rule should be removed from the episodic memory.

In FVPS, if the agent finds the same rule as the ex-

begin
Initialize $W(x,a)$ arbitrarily and TotalSteps=0.
Repeat (for each episode n){
do {
- get sensory input x_t of the environment ;
- choose a_t from an available action set A_t ;
- take action a_t ;
- *FirstVisit Routine (episodicMemory[], currentStackSize, (x_t, a_t))* ;
- check reward r_t ;
- TotalSteps ++ ;
} while (reward ! = 0) ;
if $r_t = R(>0)$ at time T steps then, T=TotalSteps;
$\forall(x,a)$ in the episodic memory

$$W_{n+1}(x,a) \Leftarrow W_n(x,a) + R \cdot \beta^T \tag{4}$$

} until enough number of episodes.
end

FirstVisitRoutine(episodicMemory[], currentStackSize,(x_t, a_t)):
begin
initialize pointer pt=0,
do from the first rule of episodicMemory[]{
search same rule as (x_t, a_t).
(compare (x_t, a_t) with stacked rule$(x,a)_{pt}$ of episodicMemory[]; pt++;)
if (x_t, a_t)== episodicMemory[pt=k], break;
} while ($pt == currentStackSize$);
if found the same rule in the episodicMemory
($pt < currentStackSize$) {
retain whole rules of episodic memory,
do not stack executed rule, (x_t, a_t).
}
else {
stack executed rule, (x_t, a_t) into episodicMemory[].
currentStackSize ++;
}
end

Figure 3: FVPS algorithm

ecuted rule (x_t, a_t) in the episodic memory, it does not stack to the episodic memory. Finally, each executed rule appears only once in the episodic memory as shown in Fig. 4(2). Even in this method, assignments will be done on needless rules, such as $(10, Up)$ in maze (C). But FVPS can eliminate these needless rules by our reinforcement function, $R \cdot \beta^{TotalSteps} (0 < \beta < 1)$, in which the value is small when the length of the agent's route is long (i.e., the size of *TotalSteps* is large). Therefore, the assignment quantity on needless rules (e.g., $(10, UP)$ in maze (C)) will be smaller than the one on effective rules (e.g., $(10, Down)$) that make the length of the agent's route short, thereby causing only the effective rules to be retained. On the other hand, because both $(1, Up)$ and $(1, Down)$ in maze (C) are necessary to reach the goal, both rules are retained

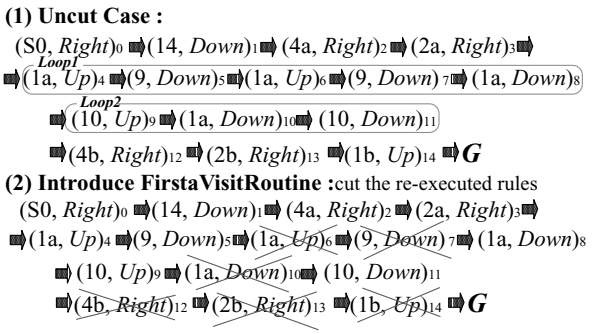


Figure 4: Example: How to Discard Looping Behavior in Maze (C)

as effective rules. Also, if there exist shorter routes to the goal without stochastic policies, the agent exploits these deterministic policies and reinforcement of the rules for the perceptual aliasing area is precluded. (We describe this using a concrete example in Section 4.2.) We claim that this approach brings about the effective stochastic or deterministic policy which is appropriate for the environment.

FVPS is very similar to the *First-visited* Monte-Carlo method[Singh & Sutton, 1996], where the assignment will be given only to the first visited state. However, the values in the Monte-Carlo method are estimated as sample averages of observed reward using the Widrow-Hoff rule (Eq. 5)[2], whereas FVPS just piles weight on successful rules according to trial and error experiences, as shown in Eq.4. It is important that the averages of observed reward among the aliasing states will fluctuate from episode to episode, and will make no sense as the value of the observation. One the other hand, piling up weights on successful rules is simple and makes agents robust in POMDPs as well.

$$\begin{aligned} \text{NewEstimate} &\Leftarrow \text{OldEstimate} \\ &+ \ \textit{StepSize}[\text{Target-OldEstimate}] \end{aligned} \tag{5}$$

4 Experiments

4.1 Settings

To demonstrate the effectiveness of the FVPS approach, we compared its performance with that of the Sarsa(λ) algorithm on the MDP(maze (A)) and five POMDP problems. (Two of them are taken from [Loch & Singh, 1998].) We describe aspects of the empirical results here. In the cases of maze (A) to maze (D), the agent starts from state S_0, as shown in

[2] *StepSize* is represented by α in Eq.1 and Eq.2.

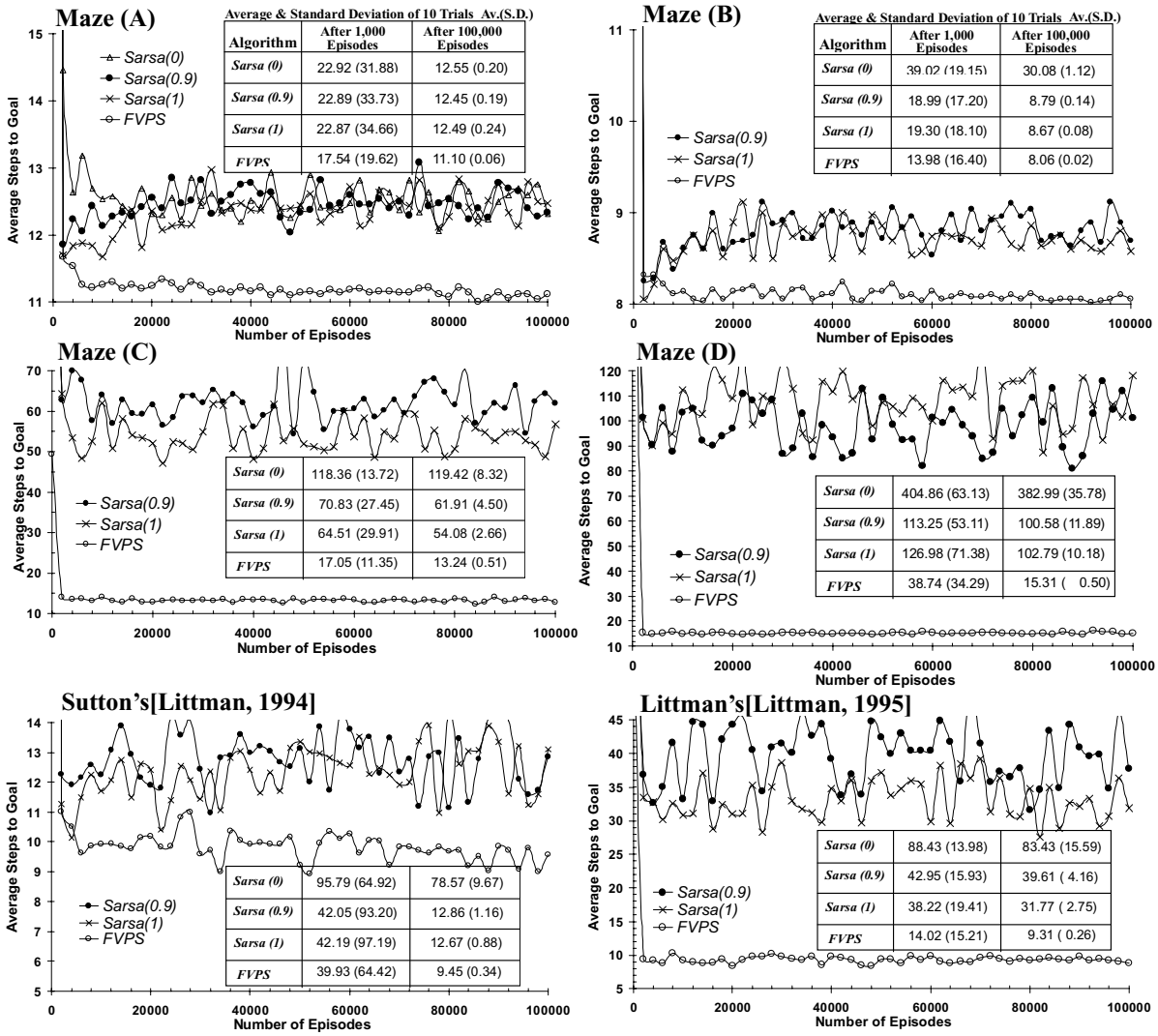


Figure 5: Comparisons among FVPS and Sarsa(λ)

Fig. 2, while in the cases of Sutton's and Littman's, it starts from different locations. At each time step, the FVPS agent selects an action by a roulette wheel method, where the rate of each action is $p(a_i|x) = \frac{W(x,a_i)}{\sum_{a_k \in A_t} W(x,a_k)}$, while the Sarsa agent uses the Boltzmann distribution $p(a_i|x) = \frac{e^{Q(x,a_i)/T}}{\sum_{a_k \in A_t} e^{Q(x,a_k)/T}}$ ($T = 0.2$) to select its action. There are four actions within the action set, $A_t = \{Up, Right, Down, Left\}$, except in Littman's case. Littman's permits an additional action, $\{Stay\}$. The weight (or value) of the rules was initialized to 10.0 in FVPS and 0.0 in Sarsa. The reward 1000.0 in FVPS and 1.0 in Sarsa is given after achieving the goal. In the Sarsa case, 0.0 is used to update at each time step. In all but Littman's case, the agent can observe its eight neighboring cells and it cannot see the goal as an entity, which means that the agent gets the reward when it is in the goal position. In Littman's case, the agent can observe the relative four directions: front, back, left and right, and can see the goal if the goal is in its relative four directions, although the observations are noisy, with the agent getting the correct observation only 70% of the time.

The parameters R, β, and T of FVPS are used to update the weight of each rule in the episodic memory. In our experiments, the rules which are retained after the *FirstVisit* routine will be given the value $R \cdot \beta^T = (1000.0) \cdot (0.8)^{TotalSteps}$. The parameters α, λ, and γ of Sarsa are also used to update the Q-value of each rule. In our experiments, both the step-size α and the λ values are held constant in each experiment. We followed Loch[Loch & Singh, 1998] to select

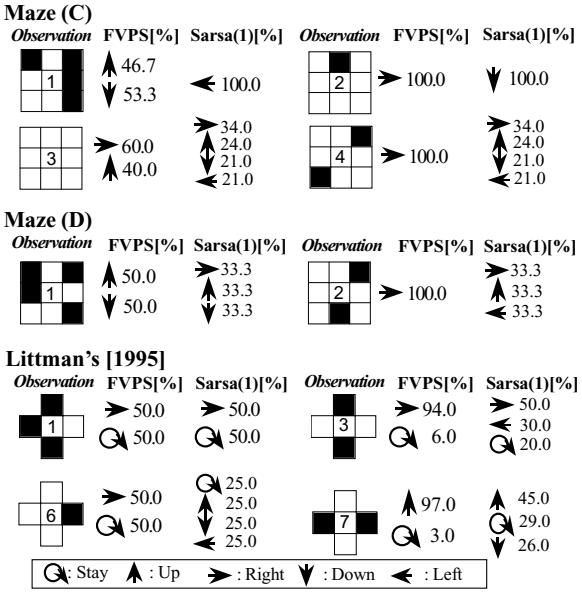


Figure 6: Policies of FVPS and Sarsa(λ) in Uncertain Areas

these values: $\alpha = 0.01$, $\lambda = 0.9$ and 1.0. We searched over γ value for these problems and selected $\gamma = 0.8$, which gave the best performance across all problems. The evaluation metric is determined by averaging the number of steps to reach the goal. Experiments consist of 10 trials, each of which consists of 100,000 episodes. The lookup table is reset for each trial.

4.2 Results

After every episode, the policy (which is selected by a roulette method in FVPS and by the Boltzmann distribution in Sarsa as mentioned in Section 4.1) was evaluated and the learning curves of all types of mazes shown in Fig. 2 plotted as shown in Fig. 5. The x-axis shows the number of episodes and the y-axis shows the average steps to the goal of 10 trials.

The maze (A) is an MDP, where Q-learning and any other DP-based algorithms can reach the optimal policy of 11 steps. Although Sarsa also can reach the optimal theoretically, it seems that step-size parameter α needs to be adjusted to reach it, because Sarsa($\lambda = 0, 0.9, 1$) starts to approach the optimal but then gets farther away from it as episodes are repeated. FVPS acquired the optimal policy only with the *FirstVisit* routine and our reinforcement function, $R \cdot \beta^{TotalSteps}$.

The maze (B) includes confusion type (B), where Q-learning and Sarsa(0) are no longer useful because of their value estimation method as described in 3.1. FVPS and Sarsa($\lambda = 0.9, 1$) could reach optimal policy here, although parameter adjustment seems to be required for Sarsa.

The maze (C) includes confusion type (C), where there are two routes to reach the goal. One route consists of 9 steps in total, $S0 - 14 - 4a - 2a - 1a - 10 - 4b - 2b - 1b - G$, and the other consists of 11 steps in total, $S0 - 14 - 4a - 5a - 3a - 10 - 4b - 5b - 3b - 6 - 1b - G$. The former route is the optimal one, but to realize it, the agent must select *Down* in $1a$ and *Up* in $1b$, whereas in the latter one the agent does not need a stochastic policy. Sarsa($\lambda = 0.9, 1$) dropped down here, because its value estimation failed even though it uses replacing eligibility traces when one observation requires different actions. As shown in Fig. 6, FVPS reached the stochastic policy in observations 1 and 3, and in 2 and 4, the agent acquired the deterministic policy. That is, FVPS acquired the effective policy that the agent needs.

The maze (D) includes confusion type (D), where there is no route to reach the goal with only a deterministic policy and where a stochastic policy is required. The Littman's maze also has this confusion because of the noise with the agent's observation. In these environments, FVPS works best and can reach nearly optimal policy within the agent's perceptual ability.

5 Discussion

The results demonstrate that FVPS found the proper policies even in the POMDPs where the currently evaluated Sarsa(λ) does not show good results. FVPS performs much better than Sarsa in the environment where more than one action will be reinforced due to the aliasing, such as mazes of (C), (D) and Littman's. In these environments, the update method of Sarsa(λ), in which the values are estimated as sample averages of observed reward by Eq.5, seems not to work well, because there is no unique value to be estimated as the rule value. FVPS, however, just piles the weight on successful rules according to the agent's trial and error experiences. This simple method seems to work very effectively in such environments.

We claim that FVPS requires an episodic memory of moderate size. Only rules which are retained in the episodic memory would be updated after each episode. Sarsa($\lambda > 0$), on the other hand, requires memory to keep eligibility traces and computation time to update all rules at each time step. The size of state spaces of environments such as those treated in this paper is very small, so the computation time for updating is not substantial. But the larger the state space, such

as in the multi-agent learning environment, the more serious a problem it might be in practical use.

6 Conclusion

In this paper, we introduce FVPS, a variant of the Profit-Sharing algorithm, and demonstrate its effectiveness within some interesting subclasses of POMDPs and its minimal memory requirements. We believe that our method will be easily introduced into a classifier system, and can solve over many subclasses of POMDPs by combining with a certain genetic algorithm. In future work, we will prove the effectiveness of FVPS theoretically and show the powerful results on more difficult classes.

Acknowledgement
This research has been sponsored in part by ONR grant N-00014-96-16-1-1222 by DARPA grant F-30602-98-2-0138.

References

[Barto et al., 1995] Barto, A.G., Bradtke, S.J., and Singh, S.P. Learn to Act using Real-Time Dynamic Programming. *Artificial Intelligence*, Vol.72, Number 1-2, pages 81-138, 1995.

[Chrisman, 1992] Chrisman, L. Reinforcement learning with perceptual aliasing: The Perceptual Distinctions Approach. *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 183-188, 1992.

[Dorigo & Bersini, 1994] Dorigo, M. and Bersini, H. A Comparison of Q-learning And Classifier Systems. *Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior*, pages 248-255 1994.

[Grefenstette, 1988] Grefenstette J. J. Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms, *Machine Learning*, Vol.3, pages 225-245, 1988.

[Hansen, 1998] Hansen, E.A. Solving POMDPs by searching in Policy Space. *Proceedings of 14th International Conference on Uncertain Artificial Intelligence*, 1998.

[Holland, 1986] Holland, J. H. Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Sysems. In R.S.Michalsky et al. (eds.), *Machine Learning: An Artificial Intelligence Approach, Vol.2*, pages 593-623, Morgan Kaufman 1986.

[Jaakkola et al., 1994] Jaakkola, T., Singh,S.P. and Jordan, M.I. Reinforcement Learning Algorithm for Partially Observable Markov decision Problems. *Advances in Neural Information Processing Systems 7*, pages 345-352, 1994.

[Lanzi, 2000] Lanzi, P.L. Adaptive Agents with Reinforcement Learning and Internal Memory. *Proceedings of 6th International Conference on Simulation of Adaptive Behavior*, pages 333-342, 2000.

[Lanzi & Wilson, 2000] Lanzi, P.L. and Wilson, S.W. Toward Optimal Classifier System Performance in Non-Markov Environments. *Evolutionary Computation*, Vol.8(4), pages 393-418, 2000.

[Loch & Singh, 1998] Loch, J. and Singh, S.P. Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes. *Proceedings of 15th International Conference on Machine Learning*, 1998.

[McCallum, 1995] MacCallum, R. A. Instance-Based Utile Distinctions for Reinforcement Learning with Hidden State. *Proceedings of 12th International Conference on Machine Learning*, pages 387-395, 1995.

[Miyazaki & Kobayashi, 1999]
Miyazaki, K. and Kobayashi, S. Proposal for and Algorithm to Improve a Rational Policy in POMDPs. *IEEE International Conference on Systems, Man, and Cybernetics*, pages 285-288, 1999.

[Moriarty et al., 1999] Moriarty, D.E., Schultz A.C. and Grefenstette J.J. Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research*, Vol.11, pages 241-276, 1999.

[Peshkin et al., 1999] Peshkin, L., Meuleau N., and Kaelbling L. Learning Policies with External Memory. *Proceedings of 16th International Conference on Machine Learning*, pages 307-314, 1999.

[Singh et al., 1994] Singh, S.P., Jaakkola, T. and Jordan, M.I. Learning Without State-Estimation in Partially Observable Markovian Decision Processes. *Proc. of the 11th International Conference on Machine Learning*, pages 284-292, 1994.

[Singh & Sutton, 1996] Singh, S.P. and Sutton, R.S. Reinforcement Learning with Replacing Eligibility Traces. *Machine Learning*, Vol.22 :1-37, 1996.

[Watkins & Dayan, 1992] Watkins, C. J. H., and Dayan, P. Technical note: Q-learning. *Machine Learning*, Vol.8: 55-68, 1992.