To appear in Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence

Experience-based Reinforcement Learning to Acquire Effective Behavior in a Multi-agent Domain

Sachiyo Arai[1], Katia Sycara[1] and Terry R. Payne[1]

The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213 USA
Phone +1 (412) 268 7019
E-Mail: {sachiyo, katia, terryp}@cs.cmu.edu

Abstract. In this paper, we discuss *Profit-sharing*, an experience-based reinforcement learning approach (which is similar to a Monte-Carlo based reinforcement learning method) that can be used to learn robust and effective actions within uncertain, dynamic, multi-agent systems. We introduce the *cut-loop* routine that discards looping behavior, and demonstrate its effectiveness empirically within a simplified *NEO (non-combatant evacuation operation)* domain. This domain consists of several agents which ferry groups of evacuees to one of several shelters. We demonstrate that the *cut-loop* routine makes the Profit-sharing approach adaptive and robust within a dynamic and uncertain domain, without the need for predefined knowledge or subgoals. We also compare it empirically with the popular Q-learning approach.

1 Introduction

Many existing approaches that reason about agent interaction have used a symbolic representation within multi-agent planning domains [5], and within the context of dynamic domains [4]. These approaches normally adopt a top-down strategy, and hence require an explicit model of the environment and a definition of the communication protocol used for multi-agent cooperation. Although the corresponding agents work successfully in complex, dynamic domains, it can be difficult to design whole parts of the agent's knowledge. As the number of agents within these multi-agent communities rises, it is becoming increasingly difficult to design static knowledge.

For dynamic domains (such as the one presented in this paper), it is not unreasonable to design agents that use local *condition-action rules* to react to each world state [4], as it can be very difficult to model the whole domain. The problem therefore becomes that of determining how these rules should be designed for dynamic environments. In recent years, bottom-up approaches such as reinforcement learning have become increasingly popular for determining these condition-action, or state-action rules, without having a priori models of the environment. However, there are still several important issues that arise when applying these bottom-up approaches to multi-agent domains.

In this paper, we present an approach known as *Profit-sharing* that allows agents to learn effective behaviors from their experiences within dynamic environments, where the agents are competitive and may have to face resource conflicts. A dynamic domain based on a *NEO (non-combatant evacuation operation)* is described, which presents the agents with limited resources and introduces uncertainty. Thus it can be very difficult to plan the different agent's activities, such as path planning and resolving resource conflicts. We demonstrate empirically that our Profit-sharing approach is effective within this domain and clarify some of the requirements that face multi-agent reinforcement learning problems.

In Section 2, we describe a simplified NEO domain from the perspective of a reinforcement learning approach, and present our agent model. Section 3 introduces the principles of Profit-sharing, the *Rationality Theorem*, which makes Profit-sharing powerful, and its advantage over other learning algorithms which are usually found within multi-agent domains. An empirical comparison of the performance of multiple agents using two learning approaches: Profit-sharing and Q-learning, is presented via several experiments in Section 4. Finally, we discuss the applicability and effectiveness of the Profit-sharing based method for real-world dynamic domains, and summarize our future work.

2 Problem Domain

Non-combatant evacuation operations, or *NEOs*, have been used to test a variety of coordination strategies. Though real-world NEOs have many constraint and resource conflicts, the domain used in this study models multiple transportation vehicles which transfer groups of evacuees to safe shelters. Each transport is operated asynchronously by an autonomous agent, which makes its own decision based on locally available information.

This NEO domain is an example of one that exhibits the following characteristics. First, there are several agents which are all "self-interested"; i.e. they pursue their own goals competitively rather than cooperatively. Second, the agents must resolve conflicts due to shared resources. Third, the agents should behave rationally, even though the domain is uncertain. By "rational", we mean that each agent should reach one of the safe shelters in a finite time period. Fourth, the domain is both uncertain and dynamic. Fifth, the agent should learn "on-line", i.e. it should learn while executing some action. Because of these characteristics, it is very difficult to design rules through mathematical analysis, as the information required by each agent is not only distributed but also changes over time.

2.1 The NEO Domain

The NEO domain consists of a grid world with multiple transporter agents, each of which carries a group of evacuees. The goal of a transporter agent is to ferry its group to one of the shelters as quickly as possible. However, there may be conflicts, as transporters cannot co-exist in the same location at the same time

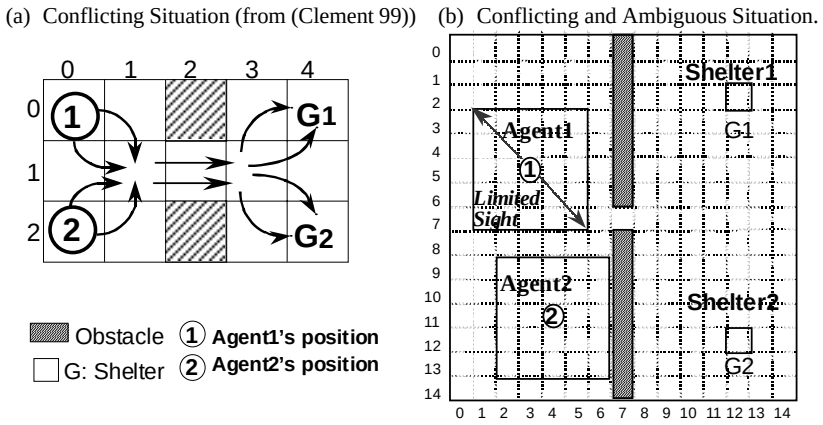


Fig. 1. Two Agents move within the grid world. Fig.(a) has been reproduced from [2]

(Figure 1a). In addition, the location of the shelters changes over the time. In dynamic domains such as this, agents should exhibit reactive behaviors rather than deliberative ones. We claim that the only effective approach is to learn reactive behaviors through trial and error experiences, since it is very difficult to know in advance what effective action should be taken at each possible state of the environment.

2.2 Modeling

Each transporter agent is modeled as a reinforcement learning entity in an unknown environment, where there is no communication with the other agents, and there are no intermediate subgoals for which intermediate rewards can be given. Thus, no reward is generated until the agent reaches its target shelter. It should be noted that there are other agents within the environment that are also learning independently of each other, without sharing sensory inputs or policies. As a result, the other agents appear as additional components within the environment, whose behavior is dynamic and unpredictable.

Each agent consists of five modules (Figure 2); a *State Recognizer*, a *LookUp Table*, an *Action Selector*, an *Episodic Memory* and the *Learner*, which includes the Profit-sharing algorithm. Initially, the agent observes O_t, the partially available state of its environment at time t. An action is then selected (using a *Roulette Selection* method) from the action set A_t, which contains all the available actions at time t. After the action is selected, the agent determines if a reward has been generated. If there is no reward after action a_t, the agent stores the state-action pair, (O_t, a_t), in its *Episodic Memory*, and repeats this cycle until a reward is generated. The terms "state-action pair" and "rule" are used interchangably in this paper. The process of moving from a start state to the

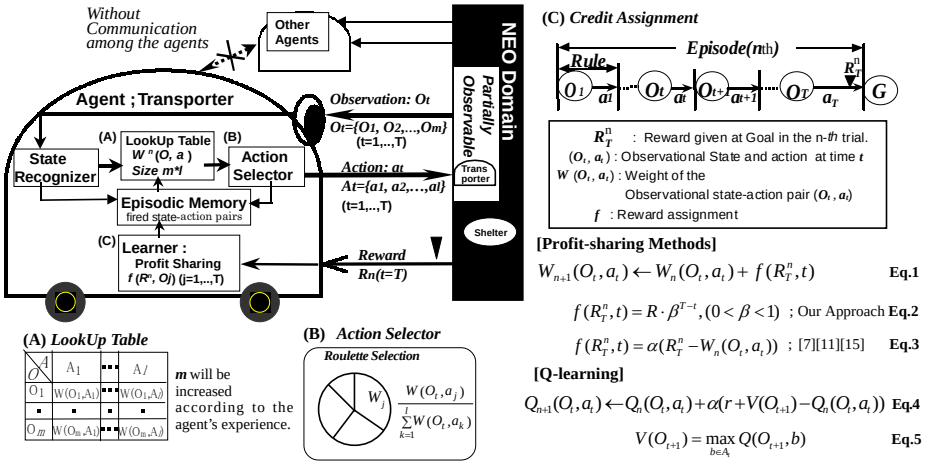


Fig. 2. Model of a Profit-sharing Agent and Credit Assignment Functions

final reward state is known as an *episode*. Once the agent receives the reward R, it reinforces the rules stored in its episodic memory by modifying the lookup table using the credit assignment function $f(R,t) = R \cdot \beta^{T-t}$ (Figure 2, Eq.2), in which $\beta(0 < \beta < 1)$ is a *discount rate*, to acquire an effective policy.

2.3 Requirements of Multi-agent Reinforcement Learning

There are three problems which have previously been encountered when reinforcement learning approaches are applied to domains with the same characteristic as our NEO domain. The first is due to the "agent's sensory limitation", in which the agent is fooled into perceiving two or more different states as the same state. This is known as *perceptual aliasing* [17]. If all these different states require the same action, then perceptual aliasing is desirable, as it results in a generalization of the state space. However, if each state requires a different action, then this can lead to the agent becoming "confused", and hence performing the wrong action. The second problem is due to *concurrent learning* [12, 1], in which the dynamics of the environment vary unpredictably as, due to learning, each agent modifies its own policies and behaviors asynchronously. Thus, midway though the learning process, an agent cannot estimate the model of state transitional probabilities for its environment. These two problems can result in non-Markovian properties within state transitions. The third problem is that the approach should minimize the amount of memory required to make an agent behave effectively.

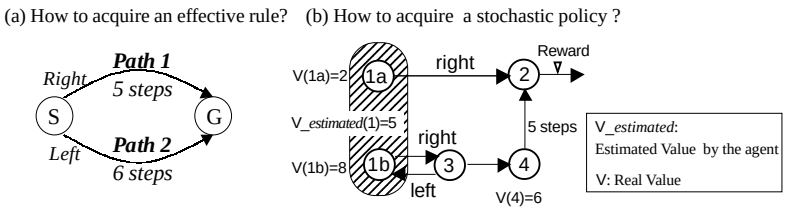


Fig. 3. Examples: (a) explains the effectiveness of β and (b) illustrates confusion in the perceptual aliasing area

3 Approach

3.1 Profit-sharing with Cut-Loop Routine

Our multi-agent reinforcement learning approach is based on *Profit-sharing*, a type of reinforcement learning originally proposed by [6]. The original version used Profit-sharing as a credit assignment method based on trial-and-error experiences, without utilizing any form of value estimation. However, this approach does not take the infinite loops in the agent's episode into consideration. These loops may result in the agent exhibiting irrational behavior with respect to achieving its goal. Although in general, the acquired policy need not be optimal for multi-agent situations, it is important that this policy is rational. A rational 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 to a rational policy in a non-Markovian domain, we introduce the *cut-loop* routine and credit assignment function, with the *discount rate* $(0 < \beta < 1)$, and describe how these augment the Profit-sharing method.

Though the *Rationality Theorem*[11] can be used to design a credit assignment function that excludes the loops without our *cut-loop* routine, applying this theorem becomes problematic when the length of the episode is long. In addition, though the *Rationality Theorem* guarantees that the ineffective rules, which make up the loop, are always given smaller rewards than the effective rules, these smaller rewards decrease the efficiency of the convergence. On the other hand, our *cut-loop* routine prohibits the agent from reinforcing the weight of the rules which make up the loop, and can shorten the length of the episode.

The function that assigns a reward among rules in the episode is called a *credit assignment function*, $f(R_t, t)$ (Figure 2, Eq.1) which denotes a reinforcement value for the rule which is fired at time t. In our Profit-sharing algorithm, the weight of each rule is reinforced according to its distance from the goal. For example, at time t, an agent enters state o_t and selects action a_t, and continues this cycle until it receives a reward R at time T. At this point, the episode consists of the rules $((o_t, a_t), (o_{t+1}, a_{t+1}), \cdots\cdots, (o_T, a_T))$, as shown in Figure 2. Each rule is then assigned some credit, according to the function $f = R \cdot \beta^{T-t} (0 < \beta < 1)$. Thus, the last rule, (o_T, a_T) is assigned credit R; the penultimate state,

(o_{T-1}, a_{T-1}), is assigned credit $R \cdot \beta$, and so on. The weight of each rule within the episode is modified by Eq.1 in Figure 2. There are two important points to note here: the weight of o_{t+1} is not required when modifying the weight of o_t; and the discount rate β assigns a greater individual reward to the most effective action than to other alternative actions. For example, consider the state diagram illustrated in Figure 3(a). If the agent's 1st(*Path 1*) and 2nd(*Path 2*) episodes required 5 and 6 steps (respectively) to achieve the goal, the weight of action *Right* at the initial state S gets a larger credit, $R \cdot \beta^5 (0 < \beta < 1)$, than the action *Left*(which gets $R \cdot \beta^6$). Therefore, the agent could choose the effective action which selects the shortest path within its experience.

The Profit-sharing algorithm is different from other methods, such as Q-learning [16] and Temporal Difference Learning [15], which make the assumption that an environment can be modeled by a Markov Decision Process(MDP). Under the Markovian assumption, the agent can perceive a set S of distinct states of its environment, and has a set A of actions that it can perform. At each discrete time step t, the agent senses the current state o_t, chooses a current action a_t, and performs it. The environment responds by giving the agent a reward $r_t = r(o_t, a_t)$ and by producing the succeeding state $o_{t+1} = \delta(o_t, a_t)$. These functions δ and r are part of the environment and are not necessarily known by the agent. Domains that obey the Markovian assumption are called MDP as the functions $\delta(o_t, a_t)$ and $r(o_t, a_t)$ depend only on the current state and action.

An agent that learns using Q-learning modifies the value of the current rule, $Q(o_t, a_t)$, using a value of sequential state $V(o_{t+1})$ to estimate the current value $V(o_t)$, as shown in Figure 2, Eq.4. At each time step, the agent updates $Q(o_t, a_t)$ by recursively discounting future utilities and weighting them by a positive learning rate α. Thus, $Q_n(o_t, a_t)$ corresponds to the nth modification of Q's components, o_t and a_t. The parameter $\gamma (0 < \gamma < 1)$ is a discount parameter, and $V(o_{t+1})$ is the value of the consecutive state (as given in Figure 2 Eq.5). Therefore, if o_{t+1} is an aliasing state, the agent fails to estimate not only the value of the current rule o_t, but also the values of the following states o_{t+1} and corresponding actions. This failed estimation will then be propagated through the learning process. To illustrate this, consider the example in Figure 3(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, it will move *left* into state 3. Otherwise, the agent moves *right* into state 1. This means that the agent learns the irrational policy where it only transits between states 1b and 3.

3.2 Cut-loop Routine in Profit-sharing

Consider the state diagram illustrated in Figure 3(b). At time t, an agent starts in state 3. If it moves left, it enters state 1. It can then return to state 3 by moving

right. Thus, the agent could cycle between these two states indefinitely, before moving onto another state (e.g. state 4) which will lead to the goal (state 2). If the agent's episode consists of the rules: $(3, Left)$, $(1, Right)$, $(3, Left)$, $(1, Right)$, ..., $(3, Right)$, $(4, Up)$, $(2, Right)$, $(Goal)$, and function f(such as the constant or simple geometrical decreasing function) does not satisfy the *Rationality Theorem*[11], then the weight of $(3, left)$ will be larger than that of $(3, right)$, as the agent will have visited $(3, left)$ several times. If the *Rationality Theorem*[11] is used to design a credit assignment function that excludes these loops, then it will fail when the length of the episode is very long.

Our solution to this problem is very simple. If the current state is the re-visited one, the agent "cuts off" the rules which make up the cyclic loop from the current *Episodic Memory*. This routine does not require any knowledge other than that used by the current framework of the Profit-sharing algorithm, because this algorithm uses an *Episodic Memory* to accumulate rules until the goal is achieved. Therefore, the agent is able to tell whether the current state is the first-visited one or the re-visited one. In the case of the above example, the original sequence of the rules becomes $(3, Right)$, $(4, Up)$, $(2, Right)$, $(Goal)$ after the *cut-loop* routine is applied. Profit-sharing uses trial and error experiences, and reinforces effective rules instead of estimating values for the different state. Therefore, it uses this policy to escape states susceptible to perceptual aliasing. This property also makes the agent robust within uncertain domains, and reduces memory requirement as it only stores rules which are essential for navigating the state space. Since the NEO domain cannot be assumed to be an MDP, and since it has a very large state space which results in very long episodes before the goal is reached, an approach that combines Profit-sharing with the *cut-loop* routine is more suitable than other reinforcement learning method(such as Q-learning).

3.3 Related Work

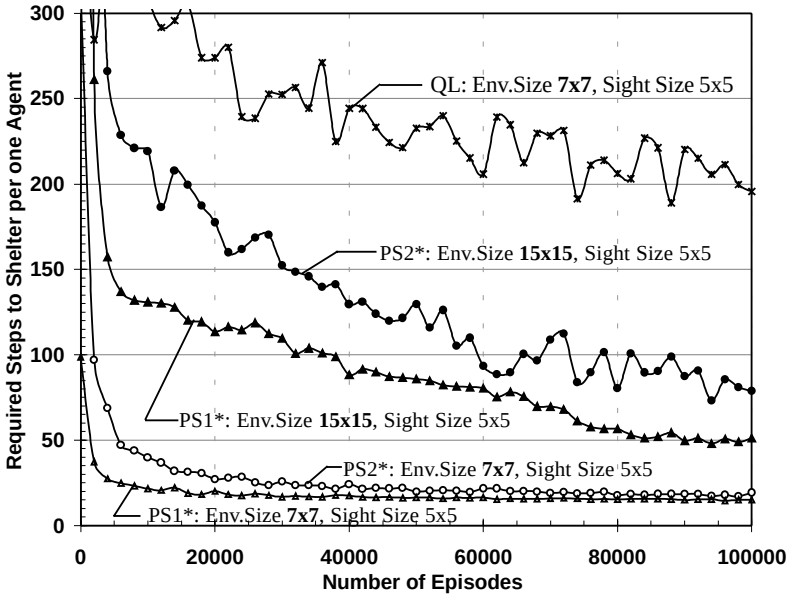
The perceptual aliasing problem has been addressed by a number of studies, and to date, two solutions have been proposed. The first is *memory-based* [3, 9], which maintains a history of rules for each episode. The second adopts a *stochastic policy* [8] where the agent selects a random action to escape from partially observable states. The first solution requires additional memory to store the tuple history. The approach adopted by our Profit-sharing algorithm is based on the later solution, which includes TD(1) and the Monte-Carlo methods [13] in that they do not use the values of consecutive states. Our approach differs from TD(1) and Monte-Carlo in that our method does not use the values of state (or state-action pairs) which require very large memory space to keep eligibility traces to manage the delayed reward. In the tabular version of TD(1) and Sarsa(1) algorithms, the required memory space is twice as large as that which is required by our Profit-sharing method.

A number of studies have recently explored the concurrent learning problem. Sub-goals were used by [10, 14] to find effective rules using Eq.3 (Figure 2), but there is no theoretical background for this approach. This problem has also been discussed theoretically for the Q-learning approach [7].

4 Experimental Results

Comparison: **PS**(*with-Cut-loop*) v.s. **PS**(*with-Rationality Theorem*) v.s. **QL**

Algorithm \ Environment Size		Average and Standard Deviation of 10 Trials				Av.(S.D.)
		After 1,000 Episodes	After 5,000 Episodes	After 10,000 Episodes	After 50,000 Episodes	After 100,000 Episodes
PS1 *Cut-Loop*	**7** x **7**	43.4(4.9)	25.1(4.1)	20.2(3.4)	16.4(1.8)	13.8(2.0)
	15 x **15**	239.3(234.8)	76.7(19.9)	70.6(21.9)	56.2(18.5)	45.5(5.7)
PS2 *with Rational f*	**7** x **7**	119.6(33.1)	50.8(15.5)	36.9(11.7)	21.6(4.7)	16.6(2.6)
	15 x **15**	642.9(297.6)	367.6(399.2)	192.4(194.8)	72.3(40.5)	69.6(40.8)
Q learning	**7** x **7**	347.7(102.1)	289.7(94.9)	297.0(90.4)	255.7(80.8)	190.6(50.7)
	15 x **15**	2135.1(665.9)	1898.4(482.8)	1716.9(453.1)	1582.5(342.9)	1260.0(306.1)



PS1* : Using Cut-loop routine and $f=(0.8)^{T-t}$
PS2* : Using Rationality Theorem and $f=(0.3)^{T-t}$, without cut-loop routine.

Fig. 4. Performance in the Dynamic and Uncertain Domain

To demonstrate the effectiveness of our Profit-sharing approach (presented in the previous Section), we compared its performance with that of Q-learning[16] on the two NEO grid worlds, as shown in Figure 1. The comparison with Q-learning is a reasonable comparison in that the memory requirements and time complexity of both algorithms are the same. (Note: Sarsa(λ) and Q(λ) need larger memory space, as mentioned in the previous Section.)

In the case of both Figure 1(a) and (b), two agents started from different locations, and their task was to learn policies for finding one of two shelters as quickly as possible. There are five actions within the action set, $A_t = \{Stay, Up, Right, Down, Left\}$. However, both agents cannot occupy the same position at the same time, nor may they pass through obstacles. In the first world (Figure 1(a); this world also appeared in [2]), the number of locations is small (5×3 locations), and the agents can see the whole environment. However, the second grid world is larger (15×15 locations), and in this case the perceptual distance of each agent is only a 5×5 region, as shown in Figure 1(b); i.e. each agent can only see a shelter or the other agent when they are no more than two moves away.

In each episode, the order in which the two agents move is determined randomly. Agents always start in the same location (i.e. $(0,0)$ & $(0,2)$ in the smaller grid world, and $(0,0)$ & $(0,14)$ in the larger one). The location of the shelters varies within the right half of the grid world in each episode. Although the first experiment may appear easier to learn, the agent will require different actions for when it occupies the left or the right half of the world. Therefore, the problem of perceptual aliasing may be greater with this than with the second experiment. When one agent reaches its target shelter, its episode terminates, and the agent remains in the shelter until the second agent reaches its goal. The evaluation metric is determined by averaging the number of states required by both agents to reach the shelters. Experiments consist of 10 trials, each of which consists of 100,000 episodes. The lookup table is reset for each trial. The learning parameters were selected as follows:

Profit-sharing: In Profit-sharing(Miya) and Profit-sharing(Ours), a geometrically decreasing function ($common\ ratio = 0.3$) and ($common\ ratio = 0.9$) was used to assign a credit to each rule, respectively. The former one satisfies the *Rationality Theorem* described above. Although the latter one does not satisfies this theorem, loops may still be removed by the *cut-loop* routine because the common ratio succeeds as the discount rate β achieves effective rules. Conflicting actions are resolved using a weighted roulette selection.

Q-learning: We used the parameters $learning\ rate = 0.05$ and $discounting\ factor = 0.9$, as these were found to be the best parameters in our experiments. When the agent reaches the goal state (i.e. the shelter), it receives a reward of 1.0. The Q-learning agent uses the Boltzmann distribution $p(a_i|s) = \frac{e^{Q(s,a_i)/T}}{\sum_{k \in actions} e^{Q(s,a_i)/T}}$ ($T = 0.2$) to select its action.

Environment 5×3 **(**without Perceptual Aliasing**)**

These experiments demonstrate the effectiveness of Profit-sharing for resolving conflicts under the concurrent learning context. The average steps-per-episode for Profit-sharing with the *cut-loop* routine, (Ours), Profit-sharing with the *Rationality Theorem*, (Miya), and Q-learning are 6.78, 6.80 and 8.98, respectively after 100,000 episodes. Initially, the difference is large, as Q-learning takes a long time to propagate the reinforcement throughout all of the rules. In contrast, Profit-sharing reinforces the successful rules immediately after one episode. Specifically, Profit-sharing(Ours) with the *cut-loop* routine converges

to the optimal policy with a smaller number of episodes(i.e. experience) than Profit-sharing with the credit assignment function which satisfies the *Rationality Theorem*. For example, the average steps per episode after 1,000 episodes for Profit-sharing(Ours) and Profit-sharing(Miya) are 14.21 and 26.70, respectively. There are some differences between the Profit-sharing methods and Q-learning towards the final stage of the experiments. This is because this environment changes in every episode and due to the concurrent learning of the agents when seeking higher rewards, and hence it is more difficult to estimate the value of the rule. Because Profit-sharing exploits successful actions in each state, its emerged plan is very closed to the optimal one. These differences are due to the concurrent learning of the agents when seeking higher rewards.

Environment $15 \times 15, 7 \times 7$ (with Perceptual Aliasing)

In these experiments, two grid worlds were used; the 15×15 world illustrated in Figure 1(b), and a similar but smaller 7×7 world. The results illustrated in Figure 4 indicate that Q-learning fails to converge for either world (only the results for the 7×7 world are shown) even after 100,000 episodes. This is not surprising, as Q-learning learns deterministic policies for MDPs, and hence is unsuited for dynamic domains. In addition, due to the perceptual limitation of the agent, the environment seems to be the non-MDPs from the agent point of view. However, Profit-sharing, which collects stochastic data and reinforces only useful rules using the *cut-loop* routine or *Rationality Theorem* could acquire an effective policy. Also in these experiments, Profit-sharing(Ours) converges to the optimal policy with smaller experience than Profit-sharing with the credit assignment function which satisfies the *Rationality Theorem*. Because the *cut-loop* routine prohibits the agent from reinforcing the weight of the rule which makes up the loop, the agent's stochastic policy becomes accurate even in the earlier stages of learning.

5 Conclusion and Future Work

In this paper, we introduce the *cut-loop* routine, and present a variant of the Profit-sharing algorithm that guarantees convergence, and demonstrate its effectiveness within a multi-agent domain characterized by conflicting situation and uncertainty. Profit-sharing solves the problems of perceptual aliasing and concurrent learning whilst minimizing memory requirements. In addition, the *cut-loop* routine makes Profit-sharing more amenable for multi-agent domains that require a stochastic policy and in which episodes become long.

While Profit-sharing is appropriate for an episodic task where the reward is only given at the end of the goal, it is less suited for domains that include intermediate rewards. We plan to combine Profit-sharing with other bottom-up approaches, such as genetic algorithms, and with top-down approaches for real world applications.

References

1. Arai,S.,Miyazaki,K., Kobayashi,S.: Generating Cooperative Behavior by Multi-Agent Reinforcement Learning, *Proceedings of 6th European Workshop on Learning Robots* p111-120 (1997).
2. Clement, J.Bradley and Durfee, H.Edmund: Top-Down Search for Coordinating the Hierarchical Plans of Multiple Agents. *Proceedings of the 3rd International Conference on Autonomous Agents,* pp252-259 (1999).
3. Chrisman, L.: Reinforcement learning with perceptual aliasing: The Perceptual Distinctions Approach, *Proceedings of the 10th National Conference on Artificial Intelligence*, pp.183-188 (1992).
4. Firby, R.J., An Investigation into Reactive Planning in complex Domains, *Proceedings of 10th National Conference on Artificial Intelligence '87*, 202-206 (1987).
5. Georgeff, M.P.: Communication and interaction in Multi-agent Planning, *Proceedings of the 3rd National Conference on Artificial Intelligence*, pp.125-129 (1983).
6. Grefenstette, J. J.: Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms, *Machine Learning Vol.3*, pp.225-245(1988).
7. Hu, Junling and Wellman, Michael P.: Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm, *Proceedings of the 15th International Conference on Machine Learning*, pp.242-250(1998).
8. 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* (NIPS-94), pp.345-352 (1994).
9. MacCallum, R. A.: Instance-Based Utile Distinctions for Reinforcement Learning with Hidden State, *Proceedings of 12th International Conference on Machine Learning*, pp387-395(1993).
10. Mataric, J.M.: Reinforcement Learning in the Multi-Robot Domain, *Autonomous Robots 4(1)*, pp.77-83(1997).
11. Miyazaki, K., Yamamura, M. and Kobayashi, S. : On the Rationality of Profit Sharing in Reinforcement Learning, *Proceedings of the 3rd International Conference on Fuzzy Logic, Neural Nets and Soft Computing*, pp.285-288 (1994).
12. Sen, S. and Sekaran, M. : Multiagent Coordination with Learning Classifier Systems, in Weiss, G. and Sen, S.(eds.), *Adaption and Learning in Multi-agent systems*, Berlin, Heidelberg. Springer Verlag, pp.218-233(1995).
13. Singh, S.P. and Sutton, R.S.: Reinforcement Learning with Replacing Eligibility Traces, *Machine Learning Vol.22*, pp.1-37(1996).
14. Stone, P. and Veloso, M.: Team Partitioned, Opaque Transition Reinforcement Learning, *Proceedings of the 3rd International Conference on Autonomous Agents*, pp.206-212(1999).
15. Sutton, R.S.: Learning to Predict by the Methods of Temporal Differences, *Machine Learning, Vol. 3*, pp.9-44(1988).
16. Watkins, C. J. H., and Dayan, P.: Technical note: Q-learning, *Machine Learning Vol.8*, pp.55-68(1992).
17. Whitehead, S. D. and Balland, D. H.: Active perception and Reinforcement Learning, *Proceedings of the 7th International Conference on Machine Learning*, pp.162-169(1990).