International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:4, 2008

A Modular On-line Profit Sharing Approach in Multiagent Domains

Pucheng Zhou, and Bingrong Hong

Abstract**—**How to coordinate the behaviors of the agents through learning is a challenging problem within multi-agent domains. Because of its complexity, recent work has focused on how coordinated strategies can be learned. Here we are interested in using reinforcement learning techniques to learn the coordinated actions of a group of agents, without requiring explicit communication among them. However, traditional reinforcement learning methods are based on the assumption that the environment can be modeled as Markov Decision Process, which usually cannot be satisfied when multiple agents coexist in the same environment. Moreover, to effectively coordinate each agent's behavior so as to achieve the goal, it's necessary to augment the state of each agent with the information about other existing agents. Whereas, as the number of agents in a multiagent environment increases, the state space of each agent grows exponentially, which will cause the combinational explosion problem. Profit sharing is one of the reinforcement learning methods that allow agents to learn effective behaviors from their experiences even within non-Markovian environments. In this paper, to remedy the drawback of the original profit sharing approach that needs much memory to store each state-action pair during the learning process, we firstly address a kind of on-line rational profit sharing algorithm. Then, we integrate the advantages of modular learning architecture with on-line rational profit sharing algorithm, and propose a new modular reinforcement learning model. The effectiveness of the technique is demonstrated using the pursuit problem.

Keywords**—**Multi-agent learning; reinforcement learning; rational profit sharing; modular architecture.

I. Introduction

A multiagent system (MAS) in which there is a number of autonomous agents interacting, with each affecting the actions of the others essentially constitutes a complex system. Performing and completing tasks in such an environment can be extremely difficult. In this paper, the cooperative MAS is concerned, in which several agents attempt, through their interaction, to jointly solve tasks or to maximize their utility[1]. Learning enables MAS to be more flexible and robust, and makes them better able to handle uncertain and changing circumstances. Thus how to coordinate different agents' behaviors by learning so as to achieve the common goal is an important theme in multiagent domains. (e.g., see [2, 3])

Manuscript received June 26, 2006. This work was supported in part by the National Science Foundation under Grant 69985002.

Pucheng Zhou is with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, HeiLongJiang Province, P.R.China(corresponding author to provide phone: 86-451-86413388; fax: 86-451-86221048; e-mail: zhoupc@hit.edu.cn).

Bingrong Hong is with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, HeiLongJiang Province, P.R.China (e-mail: hongbr@hit.edu.cn).

Reinforcement learning (RL)[4, 5] is the problem faced by an agent that must learn behavior through trial-and-error interactions with dynamic environments, and it has been applied successfully in many single agent systems. Learning from the environment is robust because agents are directly affected by the dynamics of the environment. Because of these characters, RL has become one of the important learning approaches for multiagent learning. (e.g., see [6, 7])

In a multiagent domain, to achieve the sharing goal, learning agent needs to augment its state space to include more effective information. At the same time, the increment of state space will lead to slow down the learning process. Based on the idea of modular architecture, Ono[8] proposed a modular Q-learning model, which can reduce the state space effectively. However, Q-learning is based on the environment which should obey the Markovian property, that is, the environment can by modeled by a Markov Decision Process, but for multiagent domain these cannot be met with. The rational profit sharing, which is proposed by Miyazaki[9], is a credit assignment mechanism that bases on trials and errors. Learning agents interact with the environment through trial and error, and reinforce effective state-action pairs and suppress ineffective state-action pairs by learning episode. Profit sharing in contrast with other reinforcement learning approaches which are based on Dynamic Programming, such as Temporal Difference method and Q-learning, in that profit sharing guarantees convergence to an effective policy even in domains that do not obey the Markovian property, if a task is episodic and a credit is assigned in an appropriate manner[10, 11]. Based on these researches, in this paper we integrate the advantages of modular architecture with rational profit sharing, and propose a new modular reinforcement learning approach.

II. Modular Reinforcement Learning

A. Markov Decision Processes and Reinforcement Learning

Markov Decision Process (MDP) is generally regarded as the mathematical foundation for RL. A fully observable MDP is a quadruple (S, A, T, R), where S is a finite set of states, A is a set of actions, $T: S \times A \times S \to [0,1]$ is the state transition function that describes the probability $p(s' \mid s, a)$ of ending up in state s' when performing action a in state s, and

$R : S \times A \to \Re$ is reward function that returns a numeric value after taking action a in state s.

An agent's policy is a mapping $\pi : S \to A$. The goal of the agent is to find an optimal policy π that maximizes the expected sum of discounted rewards

$$V(s,\pi) = \sum_{t=0}^{\infty} \gamma^t E(r_t \mid \pi, s_t) \tag{1}$$

where r_t is the scalar reward at time step t, $\gamma \in [0,1]$ is discount factor. Equation (1) can be rewritten as

$$V(s,\pi) = r(s,a_\pi) + \gamma \sum_{s' \in S} p(s' \mid s, a_\pi) V(s',\pi) \tag{2}$$

where a_π is the action determined by policy π.

Dynamic Programming (DP) theory[12] guarantees there exists at least a deterministic optimal policy π^*, that for any $s \in S$, it satisfies

$$V(s,\pi^*) = \max_{a \in A}\{r + \gamma \sum_{s' \in S} p(s'|s,a) V(s',\pi^*)\} \tag{3}$$

Q-learning[13] is one kind of model-free RL algorithm is that based on DP. In essence Q-learning is a temporal-difference learning method. Q-learning directly computes the approximation of an optimal action-value function, called Q-value, by using the following update rule

$$\begin{aligned} Q(s_t,a_t) &\leftarrow Q(s_t,a_t) + \\ &(1-\alpha)\left(r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right) \end{aligned} \tag{4}$$

here $\alpha \in [0,1]$ is learning rate.

B. *Modular Q-learning*

The modular Q-learning architecture[8] consists of a mediator module and learning modules which amount to the number of agents involved in the task, shown as Fig 1. Each agent in the learning module carries out Q-learning in the environment. In each learning module, learning concentrates on a single agent and the learning of other agents is not considered. To complete the global goal, it needs a mediator module to arbitrate the results of the learning modules. The mediator module combines learning modules' decision policies using a simple heuristic decision procedure and determines a final decision by the corresponding agent.

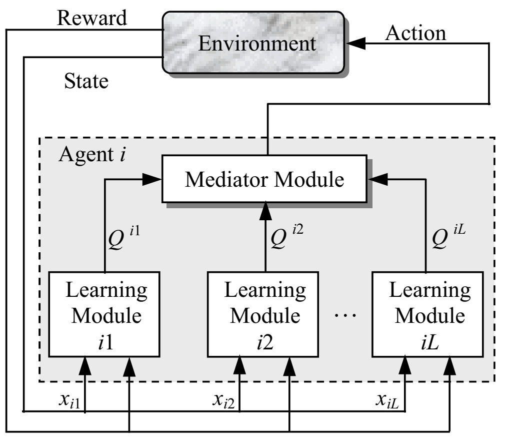

Fig. 1 The modular architecture for a learning agent

The learning agents can decide their action by using the Greatest Mass (GM) strategy proposed by Whitehead[14],

$$\arg\max_{a \in A} \sum_{k=1}^{L} Q^{ik}(x_{ik},a). \tag{7}$$

where Q^{ik} denotes the action-value function sustained by the learning module ik, L is the number of learning modules. Intuitively, this is a selection b majority among actions proposed by individual learning-modules.

III. Rational Profit Sharing

Profit sharing is originally proposed by Grefenstette[15]. 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 guarantee the rationality of an acquired policy. To guarantee convergence to a rational policy in a non-Markovian domain, Miyazaki[9, 16] introduces the rationality theorem, which the credit assignment function should satisfy, here we call it rational profit sharing.

In a multiagent environment, at time step *t* the learning agent observes a state s_t, which is generally the partially available state of its environment. An action a_t is then selected from the action set. After the action is executed, the agent determines whether a reward *r* is generated. If there is no reward, the agent stores the state-action pair so-called rule (s_t, a_t) in its episodic memory, and repeats this cycle until a reward is generated. The process of moving from an initial state to the final reward state is called an episode. When a reward is given to the agent, the rules stored in its episodic memory are reinforced at once. The function that maps states to actions is called a policy. A policy is called rational if and only if the expected reward per an action is larger than zero. We call a subsequence of an episode a detour when different actions are selected for the same state in an episode. The rules on a detour are called ineffective rules. To control the ineffective rules, Miyazaki[16] gives the following rationality theorem of profit sharing.

Lemma 1. Any ineffective rule can be suppressed iff

$$\forall i = 1,2,\ldots,W, \quad C\sum_{j=i}^{W} f_j < f_{i-1} \tag{8}$$

where C is an upper bound of the number of conflicting effective rules, and W is an upper bound of the length of episodes. In practice, we can let C=M-1, where M is the number of actions. f_i denotes a reinforcement value for the rule selected at *i* step before a reward is acquired. The inequality (8) is called suppression condition.

Lemma 2. Profit sharing can learn a rational policy iff it satisfies the suppression condition of lemma 1.

The algorithm for the profit sharing based on lookup table is followed.

Algorithm 1. The profit sharing algorithm

```
Initialize, for all $s \in S, a \in A(s)$:
    $w(s,a) \leftarrow$ a small constant
Repeat (for each episode)
    $t \leftarrow 0$, initialize $s_t$
```

```
  Repeat (for each step of episode)
    According to $s_t$, choose $a_t$ using weighted-roulette:
      $\Pr(a_t = a \mid s_t = s) = \frac{w(s,a)}{\sum_{a' \in A(s)} w(s,a')}$
    Save state-action pair ($s_t$, $a_t$) into episodic memory
    Perform action $a_t$; observe reward $r_{t+1}$, and next state $s_{t+1}$
    $t \leftarrow t+1$
  until $s_{t+1}$ is terminal state
  $T \leftarrow t$, for each pair ($s_i$, $a_i$) in episodic memory:
    $w(s_i,a_i) \leftarrow w(s_i,a_i) + f(i,r_T), i = 0,1,...,T-1$
until the algorithm holds the stopping criteria
```

IV. Modular On-Line Profit Sharing Approach

Within the modular-Q model, each learning module only concerns with a part of information of the environment, this may reduce the size of state space. However, it will easily lead to perceptual aliasing problem[17], i.e., the learning agent wrongly treat two or more different states as one. At the same time, in a multi-agent domain, the succeeded state of the learning agent will not only be effected by its action and the environment, but also be effected by other agents' behaviors, in other words, the environment cannot obey Markovian property. However, even within non-Markovian environment, profit sharing method can learn effective policy. So, here we adopt profit sharing as a basic algorithm for each learning module.

A. On-line Profit Sharing Algorithm

Profit sharing algorithm shown in Algorithm 1 is an off-line updating method, which does not change weights of its state-action pairs until the end of a task. One drawback is that it will require unbounded memory space to store all of the selected state-action pairs during the task. Based on the idea of eligibility traces[5,18], here we propose an on-line profit sharing algorithm.

The idea behind eligibility traces is very simple. Each time a state is visited it initiates a short-term memory process, a trace, which then decays gradually over time. This trace marks the state as eligible for learning. There are mainly two kinds of eligibility traces, that is accumulating and replacing eligibility traces, as defined by equation (9) and (10), respectively.

$$e_t(s,a) = \begin{cases} \lambda e_{t-1}(s,a) + 1 & s = s_t, a = a_t \\ \lambda e_{t-1}(s,a) & otherwise \end{cases} \tag{9}$$

$$e_t(s,a) = \begin{cases} 1 & s = s_t, a = a_t \\ 0 & s = s_t, a \neq a_t \\ \lambda e_{t-1}(s,a) & s \neq s_t \end{cases} \tag{10}$$

where λ is the decay parameter, and $0 \le \lambda \le 1$.

We apply eligibility traces for implementing profit sharing method incrementally, and propose a kind of on-line profit sharing algorithm, as shown in Algorithm 2.

Algorithm 2. The on-line profit sharing algorithm

```
Initialize, for all $s \in S, a \in A(s)$:
  $w(s,a) \leftarrow$ a small constant
Repeat (for each episode)
  for all $s$, $a$ : $e(s,a) \leftarrow 0$
  $t \leftarrow 0$, initialize $s_t$
  Repeat (for each step of episode)
    According to $s_t$, choose $a_t$ using weighted-roulette:
      $\Pr(a_t = a \mid s_t = s) = \frac{w(s,a)}{\sum_{a' \in A(s)} w(s,a')}$
    Update eligibility traces:
    ● if use accumulating eligibility traces, then
        $e_t(s_t,a_t) \leftarrow e_{t-1}(s_t,a_t) + 1$
    ● if use replacing eligibility traces, then
        $\begin{cases} e_t(s_t,a_t) \leftarrow 1 \\ e_t(s_t,a_t) \leftarrow 0, \text{for} \forall a \in A(s_t), a \neq a_t \end{cases}$
    Take action $a_t$; observe reward, $r_{t+1}$, and next state, $s_{t+1}$
    For all $s$, $a$:
      $\begin{cases} w(s,a) \leftarrow w(s,a) + r_{t+1} e_t(s,a) \\ e_t(s,a) \leftarrow \lambda e_t(s,a) \end{cases}$
    $t \leftarrow t+1$
  until $s_{t+1}$ is the terminal state
until the algorithm holds the stopping criteria
```

Theorem 1. If the following hold:

1. Algorithm 1 adopts credit assignment function
$$f(i,r_T) = \gamma^{T-i-1} r_T$$
where $\gamma \le \frac{1}{C+1}$, and $\gamma = \lambda$.
2. Algorithm 2 uses accumulating eligibility traces.
3. the intermediate rewards are zero until the end of each episode.

Then, Algorithm 1 is equivalent to Algorithm 2, both of them can learn a rational policy.

Proof. First, we will prove that these two algorithms are equivalent.

According to Kronecker Delta function:

$$\delta(i,j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Equation (9) can be rewritten as

$$\begin{aligned} e_t(s,a) &= \lambda e_{t-1}(s,a) + \delta(s,s_t)\delta(a,a_t) \\ &= \lambda^t \delta(s,s_0)\delta(a,a_0) + \lambda^{t-1}\delta(s,s_1)\delta(a,a_1) + \cdots \\ &\quad + \lambda\delta(s,s_{t-1})\delta(a,a_{t-1})) + \delta(s,s_t)\delta(a,a_t) \\ &= \sum_{n=0}^{t} \lambda^{t-n}\delta(s,s_n)\delta(a,a_n) \end{aligned}$$

In Algorithm 2, for any state-action pair (s, a), after one iteration the incremental weight is $\Delta w(s,a) = r_{t+1} e_t(s,a)$, so at time step T when an episode ends, the total update is

$$\begin{aligned} \Delta w_{on}(s,a) &= \sum_{t=0}^{T-1} r_{t+1} e_t(s,a) \\ &= \sum_{t=0}^{T-1} r_{t+1} \sum_{n=0}^{t} \lambda^{t-n}\delta(s,s_n)\delta(a,a_n) \end{aligned}$$

$$= r_1\lambda^{-n}\delta(s,s_n)\delta(a,a_n) + r_2\sum_{n=0}^{1}\lambda^{1-n}\delta(s,s_n)\delta(a,a_n) + \cdots$$
$$+ r_T\sum_{n=0}^{T-1}\lambda^{T-n-1}\delta(s,s_n)\delta(a,a_n)$$

Because all the intermediate rewards until the end of an episode are zero, that is, $r_1 = r_2 = \cdots r_{T-1} = 0$, so

$$\Delta w_{on}(s,a) = r_T\sum_{n=0}^{T-1}\lambda^{T-n-1}\delta(s,s_n)\delta(a,a_n) \quad (11)$$

On the other hand, for Algorithm 1, at time step *T* when an episode is over, for any state-action pair, the sum of incremental weight can be written as

$$\Delta w_{off}(s,a) = \sum_{i=0}^{T-1} f(i,r_T)\delta(s,s_i)\delta(a,a_i) \quad (12)$$

Since $f(i,r_T) = \gamma^{T-i-1}r_T$, equation (12) can be transformed to

$$\Delta w_{off}(s,a) = r_T\sum_{i=0}^{T-1}\gamma^{T-i-1}\delta(s,s_i)\delta(a,a_i) = \Delta w_{on}(s,a)$$

Therefore, for any state-action pair, the sums of all the updates in these two algorithms are same after learning, so they are equivalent.

Next, we will show that these two algorithms can learn a rational policy.

For Algorithm 1, according to the credit assignment function, we will have

$$C\sum_{i=0}^{t-1} f(i,r_T) = C\sum_{i=0}^{t-1}\gamma^{T-i-1}r_T = r_T\gamma^{T-t-1}\frac{C(\gamma-\gamma^{t+1})}{1-\gamma}$$
$$< r_T\gamma^{T-t-1}\frac{C\gamma}{1-\gamma}$$

Because

$$\gamma \le \frac{1}{C+1} \Rightarrow C\gamma \le 1-\gamma,$$

so

$$C\sum_{i=0}^{t-1} f(i,r_T) < r_T\gamma^{T-t-1}\frac{C\gamma}{1-\gamma} \le r_T\gamma^{T-t-1} = f(t,r_T).$$

Therefore, it satisfies the suppression condition in lemma 1, and according to lemma 2, it will learn a rational policy. Because of the equivalence of Algorithm 1 and 2, we can conclude that Algorithm 1 will also learn a rational policy.

Theorem 2. If the following hold:

1. $2\lambda - \lambda^W < 1$, where λ is the decay parameter in equation (10), and *W* is an upper bound of the length of episodes in Lemma 1.
2. Algorithm 2 uses replacing eligibility traces.
3. the intermediate rewards are zero until the end of each episode.

Then, Algorithm 2 can learn a rational policy.

Proof. First, define a function

$$f(i,j) = 1-\delta(i,j) = \begin{cases} 0, & i = j \\ 1, & i \ne j \end{cases}$$

Then equation (10) can be rewritten as

$$e_t(s,a) = f(s,s_t)\lambda e_{t-1}(s,a) + \delta(s,s_t)\delta(a,a_t)$$

$$= \lambda^n\{f(s,s_t)\cdots f(s,s_{t-n+1})\}e_{t-n}(s,a) + \cdots$$
$$+\lambda f(s,s_t)\delta(s,s_{t-1})\delta(a,a_{t-1}) + \delta(s,s_t)\delta(a,a_t)$$
$$= \sum_{n=0}^{t-1}\left\{\lambda^{t-n}\delta(s,s_n)\delta(a,a_n)\prod_{i=n+1}^{t} f(s,s_i)\right\} + \delta(s,s_t)\delta(a,a_t)$$

For any state-action pair (*s*,*a*), after one iteration the incremental weight is $\Delta w(s,a) = r_{t+1}e_t(s,a)$, so at time step *T* when an episode ends, the total update is

$$\Delta w_{on}(s,a) = \sum_{t=0}^{T-1} r_{t+1}e_t(s,a)$$

Because all the intermediate rewards until the end of an episode are zero, that is, $r_1 = r_2 = \cdots r_{T-1} = 0$, so

$$\Delta w_{on}(s,a) = r_T e_{T-1}(s,a)$$

Assume at time step *T*-1 the visited state-action pair is (s_c, a_d), then

$$\Delta w_{on}(s_c,a_d) = r_T e_{T-1}(s_c,a_d) = r_T$$

At the same time, for any other unvisited state-action pair at time *T*-1, which is denoted by

$$(s_u,a_v) \in U \equiv \{(s_u,a_v) \mid s_u \in S, a_v \in A(s_u)\} - (s_c,a_d)$$

we have

$$\Delta w_{on}(s_u,a_v) = r_T e_{T-1}(s_u,a_v)$$

Then for all unvisited state-action pairs at time *T*-1, the sum of incremental weights is

$$\sum_{(s_u,a_v)\in U}\Delta w_{on}(s_u,a_v) = \sum_{(s_u,a_v)\in U} r_T e_{T-1}(s_u,a_v)$$
$$= r_T\sum_{n=0}^{T-2}\lambda^{T-n-1}\sum_{(s_u,a_v)\in U}\left\{\delta(s_u,s_n)\delta(a_v,a_n)\prod_{i=n+1}^{T-1} f(s_u,s_i)\right\} \quad (13)$$
$$+ \sum_{(s_u,a_v)\in U} r_T\{\delta(s_u,s_{T-1})\delta(a_v,a_{T-1})\}$$

Because at time *T*-1 for any (s_u, a_v), it holds

$$\delta(s_u,s_{T-1})\delta(a_v,a_{T-1}) = 0$$

so

$$\sum_{(s_u,a_v)\in U} r_T\{\delta(s_u,s_{T-1})\delta(a_v,a_{T-1})\} = 0 \quad (14)$$

According to the definition, it is apparent that

$$\prod_{i=n+1}^{T-1} f(s_u,s_i) \le 1$$

so

$$\delta(s_u,s_n)\delta(a_v,a_n)\prod_{i=n+1}^{T-1} f(s_u,s_i) \le \delta(s_u,s_n)\delta(a_v,a_n)$$

For each time step at *n*=0, 1, … , *T*-2, within the set *U* there is at most one state-action pair would be visited, so

$$\sum_{(s_u,a_v)\in U}\delta(s_u,s_n)\delta(a_v,a_n) \le 1$$

Hence

$$\sum_{(s_u,a_v)\in U}\left\{\delta(s_u,s_n)\delta(a_v,a_n)\prod_{i=n+1}^{T-1} f(s_u,s_i)\right\} \le 1 \quad (15)$$

Integrate equation(14) and equation (15) with equation(13), we have

$$\sum_{(s_u,a_v)\in U}\Delta w_{on}(s_u,a_v)\le r_T\sum_{n=0}^{T-2}\lambda^{T-n-1}=r_T\frac{\lambda-\lambda^T}{1-\lambda}$$

Because

$$2\lambda-\lambda^W<1\Rightarrow\lambda-\lambda^W<1-\lambda\Rightarrow\lambda-\lambda^T<1-\lambda$$

Therefore

$$\sum_{(s_u,a_v)\in U}\Delta w_{on}(s_u,a_v)<r_T=\Delta w_{on}(s_c,a_d)$$

Therefore, it satisfies the suppression condition in lemma 1, according to lemma 2 Algorithm 2 can learn a rational policy.

B. The Proposed Modular Reinforcement Learning Model

Based on the analysis mentioned above, here we propose a modular profit sharing model, as shown in Fig 2. The learning model of the agent consists of three modules, they are State Decomposition Module (SDM), Learning Module (LM) and Mediator Module (MM), respectively. SDM determines the state of each LM, and then each LM sends its local policy to MM. MM will determine to choose an appropriate action for the learning agent to take.

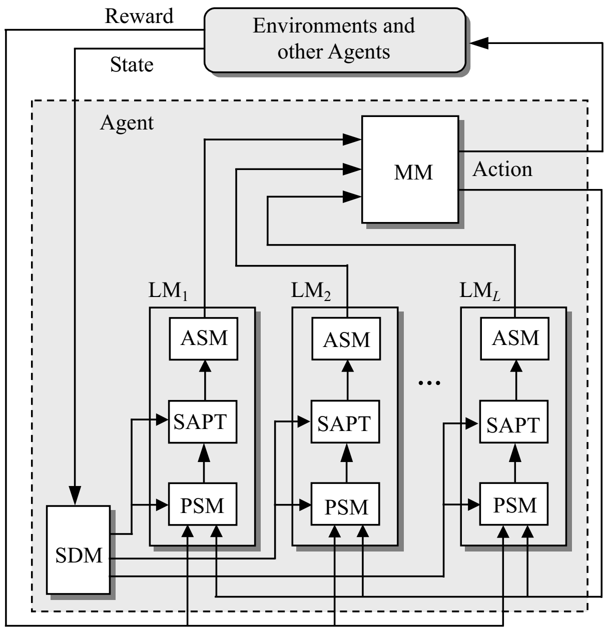

Fig. 2 The modular on-line profit sharing model of the agent

Each LM consists of three submodules.

ASM: Action Selection Module, which will send the local policy to MM, including the available actions and their weights for the current state.

SAPT: State-Action Pair weight Table, which keeps all of the state-action pairs and its weight, also it stores the eligibility trace of each state-action pair.

PSM: Profit Sharing Module. According to the received reward *r*, the current state *s* and the chosen action *a* of MM, PSM updates the weight of each state-action pair using on-line profit sharing algorithm.

The proposed learning algorithm is shown as Algorithm 3.

Algorithm 3. The proposed modular on-line profit sharing algorithm

```
1. Initialize, set learning parameters, for all $s\in S, a\in A(s)$ in each LM:
      $w(s,a)\leftarrow$ a small constant
2. Repeat (for each episode):
   1) for all $s\in S, a\in A(s)$ in each LM, $e(s,a)\leftarrow 0, t\leftarrow 0$
   2) Repeat (for each step of episode)
      (1) Observe current state $s(t)$
      (2) Determine the state of each LM, $s_i(t)$($i$=1, …,$L$, $L$ is the number of LM)
      (3) Each LM sends its available actions $a\in A(s_i(t))$ and weight $w(s_i(t), a)$ to MM
      (4) Based on received policies from each LM, MM chooses action $a(t)$ using GM strategy:
            $a(t)=\arg\max_{a_i\in A(s_{ik}(t))}\sum_{k=1}^{L}w^{ik}(s_{ik}(t),a_i)$
      (5) Each LM updates the eligibility trace of the visited state-action pair.
      (6) Take action $a(t)$; observe reward, $r(t+1)$, and next state, $s(t+1)$
      (7) For all of the state-action pair in each LM, update its weight and eligibility trace:
            $\begin{cases} w(s,a)\leftarrow w(s,a)+r_{t+1}e_t(s,a)\\ e_t(s,a)\leftarrow\lambda e_t(s,a)\end{cases}$
      (8) If state $s(t+1)$ is a terminal state, then go to step 3); otherwise, $t\leftarrow t+1$, go to step (2)
   3) If the algorithm holds the stopping criteria, then the learning process is over; otherwise, go to step 1).
```

V. Simulation

A. Problem Domain

To evaluate the learning approach proposed in this paper, we choose a modified version of the pursuit problem[19] as the test bed. In an 8×8 toroidal grid world, a single prey and four hunter agents are placed at random positions in the grid, as shown in Fig 3(a). At each time step, agents synchronously execute one out of five actions: staying at the current position or moving north, south, west, or east. The prey and the hunter cannot share a cell, but more than one hunter can be at the same position. Also, an agent is not allowed to move off the environment. Every agent can see objects at a certain distance. The distance and the cells it covers are, respectively, called the visual depth and the visual environment of the agent. An agent having visual depth *d* can see all the cells inside the $(2d+1)\times(2d+1)$ square around it. These cells are the visual environment of the agent. Each agent is assigned a unique identifier. A hunter agent can locate the relative position and recognize the identifier of any other agents in its visual environment. The prey is captured, when all of its neighbor cell are occupied by the hunters, as shown in Fig 3(b). Then all of the agents are relocated at new random positions.

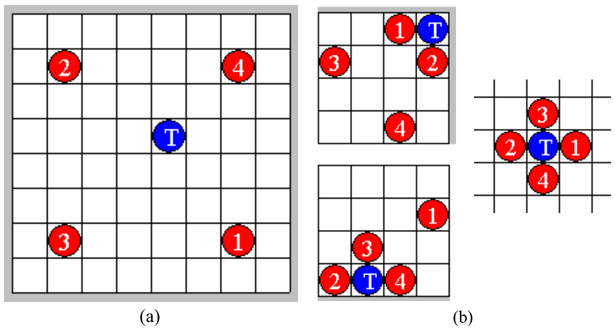

Fig. 3 (a) Example of the pursuit problem in a grid world. (b) Examples of capturing states

The pursuit problem has the following characteristics. 1) The environment is fully dynamic, partially observable and non-deterministic. 2) The hunters agents are learning agents, while the prey selects its own action based on a random strategy. 3) The hunter agents are self-interested, and there is no communication between each other.

B. Experimental Setting and Simulation Results

To test the proposed algorithm, we compare modular profit sharing based on accumulating eligibility trace (MPS-AET), with modular profit sharing based on replacing eligibility trace (MPS-RET) and Modular Q-learning (Modular-Q). According to theorem 1, in MPS-AET we set decay parameter $\lambda = 0.1$, in MPS-RET let $\lambda = 0.5$. For Modular-Q, the learning rate $\alpha = 0.1$ or $\alpha = 0.8$, discount factor $\gamma = 0.9$. The total learning episodes are 5000, and the maximum iterations within an episode are 1500 time steps. The visual depth of the hunter is d=3. Upon capturing the prey, all the hunters surrounding the prey receive a reward of 100, and accordingly all of their constituents learning modules uniformly receive the same reward regardless of what actions they have just proposed. In any other case hunters receive a reward of zero.

Since the hunter cannot finish capture the prey alone, therefore, to capture the prey, the hunter should take the information of the prey and other hunters into account, so as to better coordinate its behavior with other hunters'. For the problem of (N+1) hunters capturing one prey, according to conventional state coding, the state of Hunter$_i$ can be denoted as s:=<P, H_{i1}, …, H_{iN}>, where P is the relative position of the prey to Hunter$_i$, and H_{ik}(k=1, …, N) denotes the relative position of Hunter$_{ik}$ to Hunter$_i$. When the hunter with visual depth d, the size of the state space is $|S|=(v_s+1)^{N+1}$. For module architecture, the state space of each learning module for the hunter is a combination of the relative positions of the prey and one other hunter, that is, the size of state space is $|S_i|=(v_s+1)^2$, accordingly, the size of total state space is $|S|=N\times(v_s+1)^2$, in other words, the relationship between the size of the state space and the number of learning agents is transformed from exponential to power, which means greatly reduce the size of state space.

Fig 4 is the learning curve of the three learning algorithm under different time after several trials, Fig 5 gives the histogram of the learning results for three algorithms.

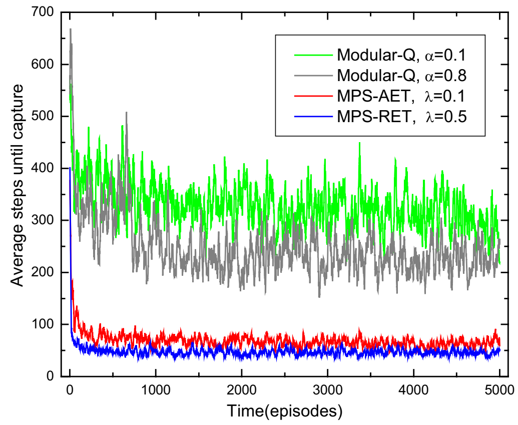

Fig. 4 The graph of comparison of three learning algorithms

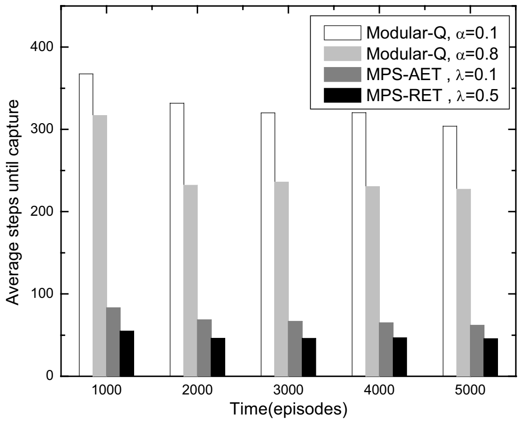

Fig. 5 The histogram of three learning algorithms

From these two graphs, we can find that, both MPS-AET and MPS-RET are better than Modular Q-learning. These can be explained as following. For the pursuit problem discussed here, the environment cannot obey Markovian property, which means the premise of Q-learning converging to the optimal policy cannot be satisfied anymore. Moreover, because of the limited sense ability, the hunter will wrongly treat two or more different states as one, which lead to the perceptual aliasing problem. Since the learning agent will choose inappropriate action, which will slow down the learning speed. In addition, Q-learning is an off-policy learning method. This means that Q-learning is guaranteed to converge to the optimal solution under MDP environment regardless of what policy is followed during training, as long as each state-action pairs is visited infinitely of in the limit. However, in Modular-Q, the one-step value updates for each learning module are computed under the assumption that all future actions will be chosen optimally for that Markovian environment. This assumption is invalid under the action selection mechanisms described above, which uses the heuristic policy such as GM strategy. Because the chosen action by mediator module will represent some compromise policy in which the different learning modules share control. This means that the computed Q-values do not converge to the actual expected return under the composite policy. Modular profit sharing method will suppress ineffective state-action pairs that may trap into loop without reward, and reinforce effective state-action pairs, so it will gradually converge to a rational policy, and obtain better learning results. Additionally, we can find that based on the parameters setting above, MPS-RET is a little better than MPS-AET, which seems someway like the reinforcement learning with replacing eligibility traces compares to reinforcement learning with accumulating eligibility traces.

To test performances of proposed algorithms under different parameter settings, we let decay parameter with different values. Fig 6 and 7 are average learning results for MPS-AET and MPS-RET under different decay parameters, respectively. When decay parameter doesn't hold the conditions of theorem 1, then the performance of the algorithm declines, the average capturing time is longer. This is because when decay parameter does not hold the conditions of theorem 1, during the learning process, some state-action pairs that will lead to trap into loop without any reward also are reinforced, so it will affect the performance of the learning algorithm. In addition, from the learning curves we can find that, different decay parameter has different effect to the learning algorithm. As for the simulation in this paper, for MPS-AET, when decay parameter $\lambda = 0.1$, the learning algorithm performs best. For MPS-RET, in generally, the algorithm performs better with the increment of decay parameters.

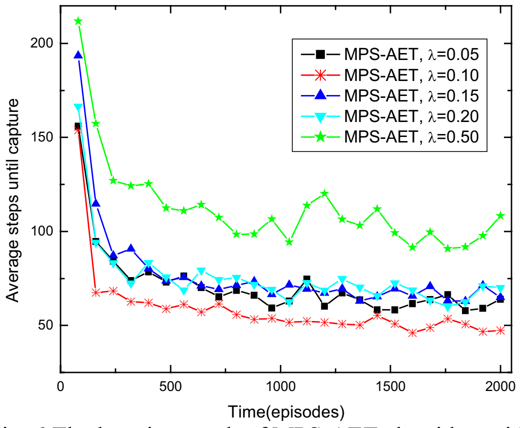

Fig. 6 The learning result of MPS-AET algorithm with different decay parameters.

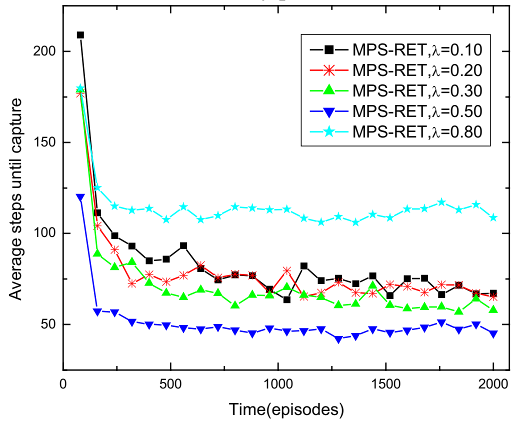

Fig. 7 The learning result of MPS-RET algorithm with different decay parameters

VI. Conclusion

In order to solve the problem of huge states in multiagent learning, as well as the perceptual aliasing problem because of the limited sense ability of learning agents, we proposed a new modular reinforcement learning approach. Firstly, a kind of on-line profit sharing algorithm is proposed, and then based on the advantages of modular architecture and on-line profit sharing method, a modular on-line profit sharing approach is presented. Experimental results showed that the proposed learning algorithm could be used to achieve optimal solution better, instead of traditional reinforcement learning methods.

References

[1] Panait L, Luke S, "Cooperative Multi-Agent Learning: The state of the art." *Autonomous Agents and Multi-Agent Systems*, 2005, 11(3): 387-434

[2] Ho F, Kamel M. "Learning coordinating strategies for cooperative multiagent systems." *Machine Learning*, 1998, 33(2-3): 155-177,

[3] Garland A, Alterman R. "Autonomous agents that learn to better coordinate." *Autonomous Agents and Multi-Agent System*, 2004, 8(3): 267-301

[4] Kaelbing L P, Littman M L, Moore A W. "Reinforcement learning: A survey." *Journal of Artificial Research*, 1996, 4: 237-285

[5] Sutton R S, Barto A G. *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 1998

[6] Excelente-Toledo CB, Jennings NR. "Using reinforcement learning to coordinate better." *Computational Intelligence*, Vol. 21 No. 3, pp. 217-245. Blackwell Publishing 2005

[7] CHEN G, YANG ZH. "Coordinating Multiple Agents via Reinforcement Learning." *Autonomous Agents and Multi-Agent Systems*, 2005, 10(3): 273-328

[8] Ono N, Fukumoto K. "Multi-agent reinforcement learning: A modular approach." *In Proceedings of the Second International Conference on Multi-agent Systems*. Portland, Oregon, USA. 1996, pp: 252-258, AAAI Press

[9] Miyazaki K, Yamamura M, Kobayashi S. "On the rationality of profit sharing in reinforcement learning." *In Proceedings of the third International Conference on Fuzzy Logic, Neural Nets and Soft Computing,* pages 285-288. Fuzzy Logic Systems Institute, 1994

[10] Arai S, Sycara K. "Effective learning approach for planning and scheduling in multi-agent domain." *In Proceedings of the 6th International Conference on Simulation of Adaptive Behavior*. Paris, France. September 2000, pp: 507-516

[11] Arai S, Sycara K P, Payne T R. "Experience-based reinforcement learning to acquire effective behavior in a multi-agent domain." *In Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*. Melbourne, Australia. 2000, pp: 125-135

[12] Bellman R. *Dynamic programming*. Princeton, NJ: Princeton University Press, 1957

[13] Watkins C J, Dayan P. "Technical Note: Q-learning." *Machine learning*, 1992, 8: 279-292

[14] Whitehead S, Karlsson J, Tenenberg J. "Learning multiple goal behavior via task decomposition and dynamic policy merging." *Robot Learning*, Norwell, MA: Kluwer Academic Press, 1993

[15] Grefenstette J J. "Credit assignment in rule discovery systems based on genetic algorithms." *Machine Learning*, 1988, 3: 225-245

[16] Miyazaki K, Kobayashi S. "On the rationality of profit sharing in partially observable markov decision process." *In Proceedings of the fifth International Conference on Information Systems Analysis and Synthesis*. Orlando, FL, USA. 1999, pp: 190-197

[17] Whitehead S D, Balland D H. Active perception and reinforcement learning. *In Proceedings of 7th International Conference on Machine Learning*. 1990, pp: 162-169

[18] Singh S P, Sutton R S. "Reinforcement learning with replacing eligibility traces." *Machine Learning*, 1996, 22: 123-158

[19] Benda M, Jagannathan V, Dodhiawalla R. "On optimal cooperation of knowledge source." Technical Report No. BCS-G2010-28, Boeing Advanced Technology Center, Boeing Computer Services, Seattle, WA, 1986