Reinforcement Learning with an Extended Classifier System in Zero-sum Markov Games

Chang Wang
College of Intelligence Science and Technology
National University of Defense Technology
Changsha, China
wangchang07@nudt.edu.cn

Hao Chen
College of Intelligence Science and Technology
National University of Defense Technology
Changsha, China
nudtchenhao15a@163.com

Chao Yan
College of Intelligence Science and Technology
National University of Defense Technology
Changsha, China
yanchao17@nudt.edu.cn

Xiaojia Xiang[*]
College of Intelligence Science and Technology
National University of Defense Technology
Changsha, China
xiangxiaojia@nudt.edu.cn

***Abstract*—A reinforcement learning (RL) agent can learn how to win against an opponent agent in zero-sum Markov Games after episodes of training. However, it is still challenging for the RL agent to acquire the optimal policy if the opponent agent is also able to learn concurrently. In this paper, we propose a new RL algorithm based on the eXtended Classifier System (XCS) that maintains a population of competing rules for action selection and uses the genetic algorithm (GA) to evolve the rules for searching the optimal policy. The RL agent can learn from scratch by observing the behaviors of the opponent agent without making any assumptions about the policy of the RL agent or the opponent agent. In addition, we use eligibility trace to further speed up the learning process. We demonstrate the performance of the proposed algorithm by comparing it with several benchmark algorithms in an adversarial soccer game against the same deterministic policy learner.**

***Index Terms*—reinforcement learning, learning classifier system, multi-agent system, markov games, eligibility trace**

I. Introduction

Multi-agent reinforcement learning (MARL) [1] remains a challenge due to the interdependency between the agents and the complexity of state-action joint representation in multi-agent systems (MAS). In this paper, we discuss the zero-sum Markov games [2] in which a RL agent tries to find the optimal policy to win against an opponent agent that is also able to learn. If the opponent agent is modeled as part of the environment, the state transition function and the reward function is non-stationary and changing over time [3]. Robot soccer game is one example of such non-stationary Markov Decision Processes (MDPs). As the policy of the opponent is usually unknown, assumptions can be made that the opponent always follows an aggressive or a conservative policy [2], [4]. However, such assumptions might be inappropriate without consideration of the opponent's actual behavior and can not guarantee the RL agent to acquire the optimal policy.

This research was supported by National Nature Science Foundation of China under grant No.61906203. Xiaojia Xiang is the corresponding author[*].

In the framework of zero-sum Markov Games, a variety of RL algorithms are based on the Minimax-Q algorithm [2], including Minimax-QS [5], Minimax-Q(λ) [6], Minimax-SARSA [6], HAMMQ [7], [8], and case-based algorithms [9], etc. These Minimax-Q based algorithms use the concept of equilibria and assume that the opponent agent always follows the optimal policy, while the Minimax-Q based agent follows a conservative policy. However, the opponent agent may also follow a suboptimal policy or other policies rather than the optimal policy, i.e., not always choosing the best action with the highest expected reward. Besides, domain knowledge has been assumed available for the choice of spreading functions or heuristics [5], [7], [8], which will influence the performance of the RL agents. These methods share the limitation of making strong assumptions about the priory information available.

It is useful for the RL agent to develop an opponent model (OM) based on the training data of the observed behaviors of the opponent. Methods such as JAL [10] and MetaStrategy have been tested in static tasks by counting the number of actions taken by other agents. Hyper-Q [11] learns the values of strategies and actions by taking other agents' model into the state-action value representation. In dynamic tasks, the OM can be constructed by calculating the frequencies of actions taken by the opponent in every state [4], [12]. In recent work, opponent modelling has been combined with deep reinforcement learning [13] as well as Bayesian policy reuse [14], [15] against the opponent agent following a non-stationary policy. However, modeling the opponent increases the representation complexity of the state-action spaces, which makes it harder to find the optimal policy.

In this paper, we take advantage of the mechanism of eXtended Classifier System (XCS) [16] to learn the action selection policy using the OM. In the literature, XCS has been used in both cooperative RL tasks [17] and competitive RL tasks [18], [19]. However, these algorithms treat the XCS agents independently without integrating the OM into the model representation of the XCS agents. In contrast, we represent the actions of the opponents and the consequent rewards as part of the XCS agent's decision model. As a result, we construct the OM from scratch based on the training data while the OM is simultaneously used to improve the action selection policy of the XCS agent. We use a population of competing rules to guide the action selection and use genetic algorithm (GA) to evolve the rules. The evolved rules are expected to be accurate with low prediction errors as well as maximally general with aggregated states. Furthermore, we propose a new method to evolve the XCS rules by matching the encountered state-action pairs with the stored data in the eligibility trace. We have evaluated the performance of the proposed algorithm in the adversarial soccer game environment Hexcer [12].

The rest of the paper is organized as follows. Section II discusses the proposed algorithm. Section III describes the experiments and results. Finally, section IV concludes the paper and outlines our plans for future work.

II. XCS WITH OPPONENT MODELLING

In this paper, we focus on the case that the opponent can learn a deterministic policy (e.g., Minimax-Q) and we leave the case of non-deterministic policy for future work. We propose an algorithm to learn the action selection policy against the opponent, combining the advantages of XCS [16], opponent modeling and Q(λ)-learning [20]. We assume that the agent can observe the states and actions of the opponent.

A. Opponent modelling

Similar to NSCP [4] that extends the modelling method in JAL [10], the XCS with opponent modelling agent also maintains a list K to record the number of times that an action o has been executed by the opponent in the state s. Then, the estimated action distribution of the opponent in s can be calculated as follows:

$$\sigma(s,o) = \frac{K(s,o)}{\sum_{o \in O} K(s,o)} \tag{1}$$

where O is the set of actions available for the opponent. After each training cycle, the agent updates the OM by recalculating the elements of K. We note that the OM is used throughout model representation, action selection and rule evolving.

B. Representation and creation of action selection rules

Similar to the original XCS [16], the agent also learns to perform a task by evolving a population [P] of action selection rules using temporal difference (TD) learning and genetic algorithm (GA). An action selection rule (also called a classifier cl in XCS) for the XCS with opponent modelling is a tuple $\langle c, a, \boldsymbol{p}, \varepsilon, F \rangle$ which contains a condition c, an action a, a payoff prediction vector $\boldsymbol{p}$, an associated prediction error ε and a fitness value F. An element $p[o]$ in $\boldsymbol{p}$ denotes the predicted payoff obtained by the agent if the agent executes the action a while the opponent selects a corresponding action o. In other words, the dimension of $\boldsymbol{p}$ is the same with the cardinal of the opponent's action set O.

The condition c of the classifier cl is represented by a vector of the ternary alphabet $\{0,1,\#\}$ in the same way as XCS, where the wildcard # can be either 0 or 1. When the agent receives an input state s from the environment, a match set [M] is formed, containing every classifier in the polulation [P] whose condition matches with s.

In the original XCS, the *covering* mechanism is necessary if the number of actions in [M] is less than a predefined threshold θ_{mna}, which guarantees the candidate actions for selection are sufficient enough. Then, new classifiers with conditions matched with s will be created for those missing actions with predefined predicted payoff p along with ε and F.

In the early stage of training, the reward from the environment can be very sparse, e.g., the agent is rewarded only when it reaches the goal state. As a result, the predicted payoff vector $\boldsymbol{p}$ and the prediction error ε of the classifiers used for action selection will be zero. According to the mechanism of XCS, the fitness F of these classifiers should be increased because they can predict the reward accurately. However, these classifiers do not contribute to the learning, and they have to be temporally kept in [P] before the experience value *exp* (used times for action selection) reaches the deletion threshold θ_{del}.

In this paper, we propose a new *covering* mechanism to decrease the influence of these classifiers. Specifically, we calculate the *expected action prediction* (EAP) for every action of the classifiers in [M] and compare it with a predefined threshold θ_{EAP}. The predicted payoff of an candidate action a_i is calculated as:

$$\boldsymbol{P}_{a_i} = \frac{\sum_{cl.a=a_i \wedge cl \in [M]} cl.\boldsymbol{p} \cdot cl.F}{\sum_{cl.a=a_i \wedge cl \in [M]} cl.F} \tag{2}$$

where $cl.\boldsymbol{p}$ and $cl.F$ denote the payoff prediction vector and the fitness value of the classifier cl, respectively. Combined with the OM, the EAP for each candidate action can be represented as:

$$\xi_{a_i} = \sum_{o \in O} \sigma(s,o) \cdot P_{a_i}[o] \tag{3}$$

If $\max \xi_{a_i} \le \theta_{EAP}$, the *covering* mechanism is evoked and new classifiers are created accordingly.

C. Action selection

Boltzmann distribution and $\varepsilon - greedy$ are two widely used action selection approaches for RL agents. $\varepsilon - greedy$ simply suggests the selection of the best action for exploitation trials and a random action in exploration trials. In our case, Boltzmann distribution seems more reasonable by maintaining a distribution of action selection probability. All candidate actions are taken into account, and the action with the higher EAP has a higher probability of being selected. To prevent the numerical issues, we choose the adapted Boltzmann distribution [3].

Different from the algorithms that select conservative actions using the minimax mechanism (e.g., Minimax-Q), our algorithm can select actions considering the OM. If the opponent never takes the action o in state s, then the payoff prediction $p[o]$ is ignored by the OM.

D. Eligibility trace

The XCS with opponent modelling algorithm maintains a trace set Ξ and an eligibility trace e to record the historical information. The trace set Ξ consists of historical state-action triples (s,a,o), where s is the input situation, a and o are the actions taken by the agent and the opponent, respectively. The eligibility trace e records the eligibility values of (s,a,o) tuples in Ξ, which determines how to update the classifiers.

After each learning cycle, the eligibility value of every tuple (s,a,o) recorded in Ξ are decreased as follows:

$$e(s,a,o) = \lambda \cdot \gamma \cdot\ e(s,a,o) \tag{4}$$

where $\gamma \in [0,1]$ is the discount factor, $\lambda \in [0,1]$ is the trace-decay parameter. We only maintains the triples whose eligibility values are significant enough. In other words, a tuple (s,a,o) will be deleted from Ξ if its eligibility value is less than the threshold θ_{et}.

We use the replacing trace technique [21] to update the eligibility trace values after the classifiers are updated. In this way, the eligibility trace e only needs to keep the non-zero values for each situation s.

E. Evolving the classifiers

Once an action a is selected, the agent forms an action set [A] containing every classifier in the match set [M] that suggests the selection of a. After a is executed, a reward r is received from the environment, which is used to update the classifiers in the previous time step's action set $[\mathrm{A}]_{-1}$ in the similar way with XCS. Besides, the action set $[\mathrm{A}]_{\mathrm{et}}$ matched with the eligibility trace is also updated.

1) Update $[\mathrm{A}]_{-1}$*:* In the RL part, the previous time step's action set $[\mathrm{A}]_{-1}$ is updated. For every classifier in $[\mathrm{A}]_{-1}$, the experience parameter *exp* is increased by 1:

$$cl.exp \leftarrow cl.exp + 1 \tag{5}$$

The target prediction P denotes the expected total discounted future payoff for $[\mathrm{A}]_{-1}$, resulting from taking the selected action a_{-1} in the previous situation s_{-1} and following the optimal policy thereafter. In other words, P is the sum of the reward r_{-1} and the discounted maximal EAP:

$$P = r_{-1} + \gamma \max_{a' \in A} \xi_{a'} \tag{6}$$

where $0 < \gamma < 1$ is a discount factor, A is the set of actions available for the agent, and $\xi_{a'}$ is the EAP for every candidate action. As the reward r_{-1} is related to the agent as well as the opponent, the prediction values of the corresponding opponent action in the payoff prediction vector $p[o]$ should also be updated. The standard Widrow-Hoff delta rule is used to update $p[o]$:

$$cl.p[o] \leftarrow cl.p[o] + \beta_1 (P - cl.p[o]) \tag{7}$$

where $0 < \beta_1 < 1$ is a learning rate.

The prediction error ε describes the difference between the target payoff after executing the action and the expected payoff using the current OM. As the OM is constantly updated

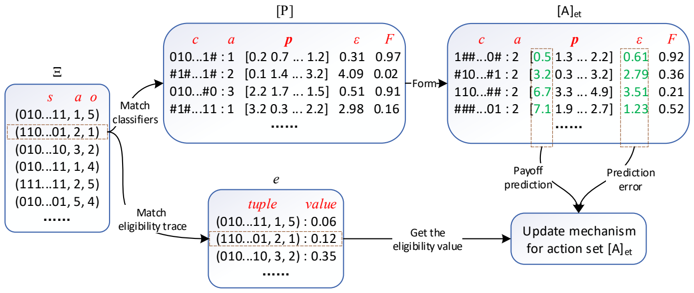

Fig. 1. The procedure of updating $[\mathrm{A}]_{\mathrm{et}}$.

during learning, the prediction error should also be updated accordingly. The *expected classifier prediction* (ECP) $cl.\xi$ denotes the expected payoff that the classifier cl receives after the agent executes the action advocated by cl using the current OM:

$$cl.\xi = \sum_{o \in O} \sigma(s_{-1}, o) \cdot cl.p[o] \tag{8}$$

where $\sigma(s_{-1}, o)$ is the OM in the situation s_{-1} of the previous time step. Then, the prediction error of each classifier in $[\mathrm{A}]_{-1}$ is updated by:

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta_2 (|P - cl.\xi| - cl.\varepsilon) \tag{9}$$

where $0 < \beta_2 < 1$ is a learning rate.

The action set size *as* and the fitness F are updated as the same way with the original XCS. After the above parameters are updated, GA and the *subsumption* mechanism are executed in $[\mathrm{A}]_{-1}$ in the same way with the original XCS.

2) Update $[\mathrm{A}]_{\mathrm{et}}$*:* Eligibility trace values represent the importance of a historical state-action triple (s,a,o) to a previous situation. For every tuple (s,a,o) in Ξ, a set $[\mathrm{A}]_{\mathrm{et}}$ is obtained by matching the situation s and the action a with the classifiers in the population [P]. Q(λ)-learning [20] updates all state-action pairs in the Q table by using the error between the target Q-value and the maximal Q-value of the current state at the previous time step. In this paper, we use this idea to update the classifiers in $[\mathrm{A}]_{\mathrm{et}}$ based on the error between the target prediction (6) and the maximum EAP in the previous time step's match set $[\mathrm{M}]_{-1}$.

$[\mathrm{A}]_{\mathrm{et}}$ is updated as shown in Fig. 1. Every tuple (s,a,o) in Ξ is matched with the classifiers in [P], and a set $[\mathrm{A}]_{\mathrm{et}}$ is formed. Meanwhile, the corresponding eligibility value for the tuple (s,a,o) is obtained from the eligibility trace. Then, the error ε and the corresponding prediction $p[o]$ of classifiers in $[\mathrm{A}]_{\mathrm{et}}$ are updated.

With the introduction of GA and the *subsumption* mechanism, the population [P] are evolving accurate and maximally general classifiers. We note that a set $[\mathrm{A}]_{\mathrm{et}}$ corresponding to a tuple (s,a,o) is not an action set, i.e., a is not actually performed before or after the update of $[\mathrm{A}]_{\mathrm{et}}$. Meanwhile, the classifier parameters of the experience *exp*, the fitness F, and the action set size *as* should be updated only when the classifier has been selected for action selection. Therefore, we only update the payoff prediction vector $\boldsymbol{p}$ and the prediction error ε for classifiers in $[\mathrm{A}]_{\mathrm{et}}$, while other parameters are not changed. Actually, we have tried to update the parameters all together, but it did not work well. Besides, the classifiers in $[\mathrm{A}]_{-1}$ will not be updated again in this process. The update rule is as follows:

$$cl.p[o] \leftarrow cl.p[o] + \beta_3 \left(P - \Phi_{-1}\right) \cdot e\left(s,a,o\right) \tag{10}$$

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta_4 \left(|P - \Phi_{-1}| - cl.\varepsilon\right) \cdot e\left(s,a,o\right) \tag{11}$$

where $0 < \beta_3 < 1$ and $0 < \beta_4 < 1$ are learning rates, and Φ_{-1} is the maximum EAP in $[\mathrm{M}]_{-1}$ calculated as follows:

$$\Phi_{-1} = \max_{a_i \in A} \sum_{o_i \in O} \sigma\left(s_{-1}, o_i\right) \cdot P\left[o_i\right] \tag{12}$$

III. Experiments and Results

We evaluate the proposed algorithm in the Hexcer environment [12], as shown in Fig. 2. Two players are located in the map made up of connected hexagonal grids. The initial states of the players and the ball are shown in Fig. 2.

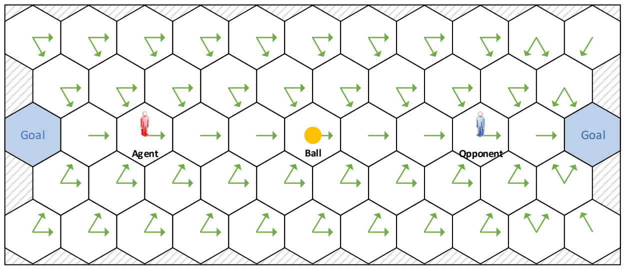

Fig. 2. The Hexcer environment and the heuristic policies (arrows) for the HAMMQ agent.

The game rules are briefly introduced as follows. There are seven actions available for each player in Hexcer: upper left, upper right, right, lower right, lower left, left, and standby. Once a player moves to the grid that has the free ball, then the ball belongs to the player until it is taken away by the opponent. If the two players meet in the same grid while one player is keeping the ball, the ball will be randomly reassigned to one of the players. If a player tries to move out of Hexcer, the move will fail and the player will stay in its grid. When one player brings the ball into the opponent's goal grid, the episode ends and the player gets a point. The game ends after a predefined number of time steps.

A. Experiment settings

In this section, we compared the performance of the proposed algorithm with a variety of benchmark multi-agent reinforcement learning (MARL) algorithms, including Minimax-Q, Minimax-QS, Minimax-Q(λ), Minimax-SARSA(λ), NSCP-z, and HAMMQ. Specifically, we chose the Minimax-QS algorithm for the opponent that can speed up Minimax-Q by using the spreading function, i.e., a single experience can update more than a single state-action pair through spatial generalization. For experimental comparison, we use NSCP-z to replace the application of NSCP in zero-sum games. If NSCP [4] is applied to deterministic policy zero-sum games, its best-response solver is nearly identical to the opponent modelling approach in [12].

The parameters of the above algorithms were basically identical to the original work [2], [5], [6], [8]. In these algorithms, the learning rates were all initialized to 1.0 and decayed at a rate of 0.9999954 for each iteration. The exploration rate was 0.2 and the discount factor $\gamma = 0.9$. Besides, the Q values of all (s,a,o) tuples and the value function $V(s)$ of all states were initialized to 0.

For Minimax-QS, the linearly decreasing spreading function was the same with [5], where $\sigma_t(s,a,o,s_i,a_i,o_i) = g_t(s,s_i) = \tau^d$, $\tau = 0.7$, and $d_{ch} = 1$. For Minimax-Q(λ) and Minimax-SARSA(λ), the trace-decay parameter $\lambda = 0.05$. For HAMMQ, $\xi = \beta = \eta = 1$, which was the same with [7], [8]. The heuristic policy provided for the HAMMQ agent and the spreading areas provided for the Minimax-QS agent are shown in Fig. 2 and Fig. 3, respectively.

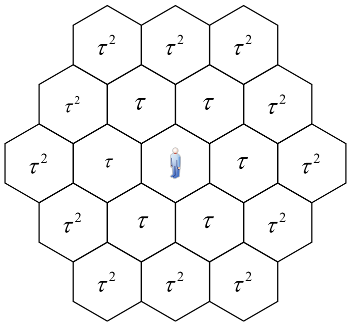

Fig. 3. Spreading areas of Minimax-QS used in Hexcer.

As can be seen from the arrows in Fig. 2, the heuristic policy requires comprehensive and accurate prior knowledge of the environment, in which the heuristic information for each state is targeted.

For our algorithm, we used a 6-bit ternary string consisting of $\{0,1,\#\}$ to encode the condition of a classifier. The population size was set to 150. We used the same learning rate $\beta_1 = \beta_2 = \beta_3 = \beta_4 = 0.15$ to update the classifiers. The mutation probability for GA was 0.2. The covering threshold $\theta_{EAP} = 0$, the eligibility trace threshold $\theta_{et} = 0.001$, and the trace decay parameter $\lambda = 0.05$. The initial prediction vector $\boldsymbol{p}$ = [0.00001 0.00001 0.00001 0.00001 0.00001 0.00001 0.00001]. Other parameter were initialized in the same way with the original XCS.

We ran a sequence of 100 experiments, and each experiment consisted of 1000 matches. Each match had 10 games, and the maximum step was 20 for each player. A player would obtain a reward of $r = 100$ if it scored one goal, otherwise $r = 0$.

B. Results

The results were averaged over 100 experiments. We note that Minimax-Q, Minimax-QS, Minimax-Q(λ), and Minimax-SARSA(λ) are abbreviated as MMQ, MM-QS, MM-Q(λ), and MM-S(λ). Besides, the proposed XCS with opponent modelling algorithm is denoted as X-OMQ(λ).

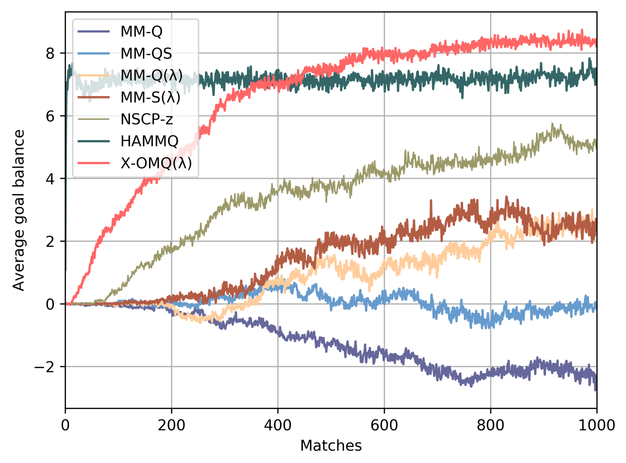

Fig. 4. Average goal balance of each match for the MARL agents against the Minimax-QS opponent in Hexcer.

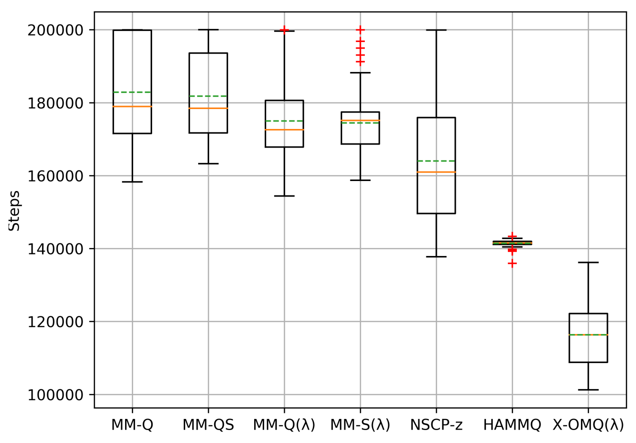

Fig. 6. Average total steps for 1000 matches of the MARL agents agaist the Minimax-QS opponent in Hexcer. The green dotted lines denote the mean values.

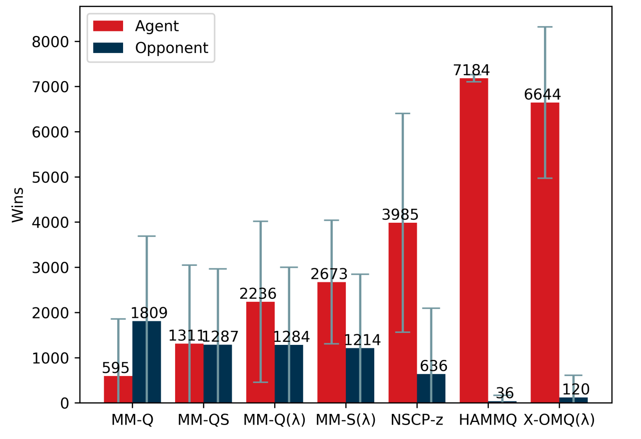

Fig. 5. Average cumulative number of wins for 1000 matches of the MARL agents against the Minimax-QS opponent in Hexcer.

Fig. 4 illustrates the average goal balance (i.e., the difference between the numbers of goal and goal lost) of each match for the agents against the Minimax-QS opponent. In this experiment, the average goal balance of the Minimax-QS agent fluctuated around 0 because the opponent used the same learning algorithm. After 200 matches, the Minimax-Q agent was completely overtaken by the Minimax-QS opponent as shown in Fig. 4 and Fig. 5. The performances of Minimax-Q(λ) and Minimax-SARSA(λ) were similar. After 1000 matches, they did slightly better than the opponent, with an average goal balance of about 2.5. Benefiting from the accurate heuristic policy, the average total steps of HAMMQ became stable over 100 experiments, as shown in Fig. 6. Our method outperformed HAMMQ after 500 matches with a smaller total steps, i.e., our method required less steps to win the opponent. In addition, we needed no heuristics as prior knowledge.

Although both the proposed algorithm and the NSCP-z used OM, our algorithm outperformed NSCP-z with the advantages of using XCS and eligibility trace for rule evolving. Our method maintained a fixed size of population while evolving the maximally general and accurate classifiers. The classifiers with low fitness values were deleted from the population, which made the action selection more reasonable and targeted for given states. Besides, once the agent received a reward from the environment, the classifiers matched in the trace set Ξ were also updated to improve the efficiency of reinforcement leaning.

C. *Results of classifiers*

Table-based MARL algorithms maintained a Q table of $53 \times 7 \times 7 = 2597$ Q-values in Hexcer. Meanwhile, our algorithm had a much less fixed population size of 150. The number of *macroclassifiers* averaged over 100 experiments was 102, and the number of selected *marcroclassifiers* in the action sets $(cl.exp > 0)$ was 72. We note that our method achieved good results only using a relatively small number of classifiers.

Table I gives examples of *macroclassifiers* with high experience $(cl.exp)$ values, i.e., frequently used for action selection. The proposed algorithm can learn the rules for specific states, e.g., the 2nd rule and 6th rule in the table. Also, it can learn rules for generalized states, e.g., the 3nd rule and 4th rule. All these rules advocate the action of moving right.

IV. Conclusion and Future Work

In this paper, we have proposed an algorithm for zero-sum Markov games. The algorithm can learn against a concurrent RL learner by constructing an opponent model from scratch, without the need of providing heuristics or assuming an agent is taking an aggressive or conservative policy as required in

TABLE I
EXAMPLE OF LEARNED CLASSIFIERS IN ONE EXPERIMENT.

No.	Situation	Action	Prediction vector	Fitness	Error	Experience	Numerosity	Action set size
1	01###1	right	[3.70 3.56 3.83 3.64 4.25 3.68 3.67]	0.09	1.88	5794	2	37.5
2	01#100	right	[4.82 4.59 4.38 4.40 4.61 4.56 4.36]	0.12	0.27	2563	2	35.0
3	##10##	right	[3.46 3.79 3.42 3.62 3.63 3.62 3.70]	0.10	0.28	2231	7	43.5
4	0###0#	right	[4.02 3.84 4.30 3.84 4.36 4.00 3.88]	0.03	2.29	2225	2	35.6
5	0#10##	right	[3.46 3.79 3.42 3.62 3.63 3.62 3.70]	0.01	0.28	1905	1	43.5
6	#11100	right	[4.82 4.59 4.38 4.40 4.61 4.56 4.36]	0.14	0.27	1833	2	35.0
7	0#10#1	right	[3.72 3.99 3.56 3.66 3.80 3.76 3.89]	0.03	0.21	1702	1	45.3
8	0110##	right	[3.46 3.79 3.42 3.62 3.63 3.62 3.70]	0.01	0.28	1643	1	43.5

the literature. Besides, the algorithm has the same advantage with XCS that maintains a population of evolving rules for searching the optimal action selection policy. As a result, most accurate and general rules can be obtained that has the representation advantage of generalizing over the state-action spaces. Different with the original XCS, the proposed algorithm deeply incorporates the opponent model throughout the procedure of rule creation, action selection and rule evolving. Furthermore, eligibility trace has been used to improve the learning efficiency. Experiment results have shown that the proposed algorithm outperforms several benchmark MARL algorithms against the same deterministic policy learner in the Hexer soccer environment. In the future work, we will further improve the proposed algorithm for non-deterministic policy learner. Besides, we will consider general sum Markov games as well, and try to solve more complex tasks in which multiple cooperative agents learn against multiple opponents that are learning concurrently.

REFERENCES

[1] L. Buşoniu, R. Babuška, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[2] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[3] H. P. van Hasselt, *Insights in reinforcement learning*. Hado van Hasselt, 2011.

[4] M. Weinberg and J. S. Rosenschein, "Best-response multiagent learning in non-stationary environments," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*. IEEE Computer Society, 2004, pp. 506–513.

[5] C. H. C. Ribeiro, R. Pegoraro, and A. H. R. Costa, "Experience generalization for concurrent reinforcement learners: the minimax-qs algorithm," in *International Joint Conference on Autonomous Agents & Multiagent Systems*, 2002.

[6] B. Banerjee, S. Sen, and P. Jing, "Fast concurrent reinforcement learners," in *International Joint Conference on Artificial Intelligence*, 2001.

[7] R. A. Bianchi, M. F. Martins, C. H. Ribeiro, and A. H. Costa, "Heuristically-accelerated multiagent reinforcement learning," *IEEE Transactions on Cybernetics*, vol. 44, no. 2, p. 252, 2014.

[8] R. A. Bianchi, C. H. Ribeiro, and A. H. R. Costa, "Heuristic selection of actions in multiagent reinforcement learning." in *IJCAI*, 2007, pp. 690–695.

[9] R. A. Bianchi and R. L. de Mantaras, "Case-based multiagent reinforcement learning: Cases as heuristics for selection of actions." in *ECAI*, 2010, pp. 355–360.

[10] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," *AAAI/IAAI*, vol. 1998, pp. 746–752, 1998.

[11] G. Tesauro, "Extending q-learning to general adaptive multi-agent systems," in *Advances in neural information processing systems*, 2004, pp. 871–878.

[12] W. Uther and M. Veloso, "Adversarial reinforcement learning," Technical report, Carnegie Mellon University, 1997. Unpublished, Tech. Rep., 1997.

[13] H. He, J. Boyd-Graber, K. Kwok, and I. H. Daum, "Opponent modeling in deep reinforcement learning," in *International Conference on International Conference on Machine Learning*, 2016.

[14] T. Yang, J. Hao, Z. Meng, C. Zhang, Y. Zheng, and Z. Zheng, "Towards efficient detection and optimal response against sophisticated opponents," *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, Aug 2019. [Online]. Available: http://dx.doi.org/10.24963/ijcai.2019/88

[15] Y. ZHENG, Z. Meng, J. Hao, Z. Zhang, T. Yang, and C. Fan, "A deep bayesian policy reuse approach against non-stationary agents," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 954–964. [Online]. Available: http://papers.nips.cc/paper/7374-a-deep-bayesian-policy-reuse-approach-against-non-stationary-agents.pdf

[16] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, Jun. 1995.

[17] K.-Y. Chen, P. A. Lindsay, P. J. Robinson, and H. A. Abbass, "A hierarchical conflict resolution method for multi-agent path planning," in *2009 IEEE Congress on Evolutionary Computation*. IEEE, 2009, pp. 1169–1176.

[18] H. Inoue, K. Takadama, and K. Shimohara, "Exploring xcs in multiagent environments," in *Proceedings of the 7th annual workshop on Genetic and evolutionary computation*. ACM, 2005, pp. 109–111.

[19] C. Lode, U. Richter, and H. Schmeck, "Adaption of xcs to multi-learner predator/prey scenarios," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010, pp. 1015–1022.

[20] P. Jing and R. J. Williams, *Incremental Multi-Step Q-Learning*, 1996.

[21] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, no. 1-3, pp. 123–158, 1996.