Non-Reciprocating Sharing Methods in Cooperative Q-Learning Environments

Bryan Cunningham
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, USA
bcunn06@vt.edu

Yong Cao
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, USA
yongcao@vt.edu

***Abstract*—Past research on multi-agent simulation with cooperative reinforcement learning (RL) focuses on developing sharing strategies that are adopted and used by all agents in the environment. In this paper, we target situations where this assumption of a single sharing strategy that is employed by all agents is not valid. We seek to address how agents with no predetermined sharing partners can exploit groups of cooperatively learning agents to improve learning performance when compared to independent learning. Specifically, we propose 3 intra-agent methods that do not assume a reciprocating sharing relationship and leverage the pre-existing agent interface associated with Q-Learning to expedite learning.**

***Keywords*-Multi-Agent Reinforcement Learning; Cooperative Learning; Agent Interaction Protocols; Information Exchanges in Multi-Agent Systems**

I. Introduction

Traditional reinforcement learning (RL) simulations imbue an agent with no knowledge of the environment in which they are located a priori. By exploring this environment and gaining direct experience with it, the agent will update its policy, or memory, to remember the best action(s) to take in each state. The goal of the agent is to improve its policy to maximize its performance in the environment [1]. Multi-agent-based cooperative RL simulations allow for multiple agents to coexist in the same environment and improve learning performance by exploiting the policy information of the other agents (Tan 1993). Enabling multiple agents to exist in the same environment adds complexity to the learning problem. Agents need to balance the tasks of exploring the environment and exploiting their own policy in addition to exploring the usefulness of the other agents' policies and exploiting them. Ideally, this improves the learning performance of the agents when compared to training them separately in single-agent RL.

Past research on multi-agent simulation with cooperative reinforcement learning (RL) focuses on developing sharing strategies that are adopted and used by all agents in the environment. We consider these sharing strategies to be reciprocating because all participating agents have a predefined agreement regarding what type of information is shared and when to share it. The sharing strategies are specifically designed around manipulating this shared information to improve learning performance. In this paper, we target situations where this assumption of a single sharing strategy that is employed by all agents is not valid. This research generalizes the cooperative learning domain by considering situations in which agents may use differing sharing strategies to cooperatively learn a task, as shown in Fig. 1. We seek to address how agents with no predetermined sharing partners can exploit groups of cooperatively learning agents to improve learning performance when compared to independent learning. Specifically, we propose 3 intra-agent methods that do not assume a reciprocating sharing relationship and leverage the pre-existing agent interface associated with Q-Learning to expedite learning.

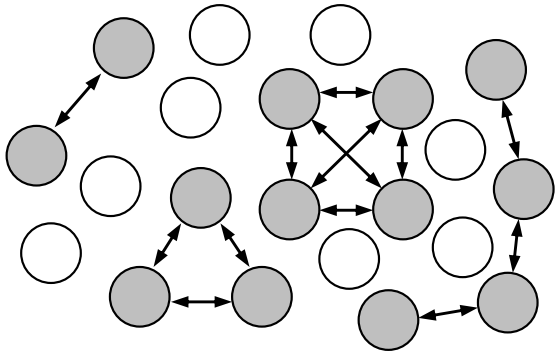

Figure 1. Separate groups of cooperative learning agents employing distinct sharing strategies interspersed with individual learning agents.

Whereas other cooperative learning research assumes the simulation designer has control of all other agents in the environment for the purposes of encoding an agent interaction strategy, this research will investigate algorithms that do not so assume. The other agents' functions and their sharing strategies are unknown and inaccessible from the point of view of the agent(s) using our proposed methods. We contribute 3 sharing methods: the modified averaging (MA) method, the modified experience counting (MEC) method, and the hybrid experience counting and averaging

(HECA) method. These 3 methods adapt existing sharing strategies to perform in environments where other agents do not reciprocate. We analyze our methods by testing them in 3 unique environments where the other agents employ a variety of standard sharing methods.

Findings indicate that learning performance while using our proposed methods is improved when compared to independent learning. Results show that agents with no predetermined sharing partners can successfully exploit groups of cooperatively learning agents. Additionally, the modified experience counting method performs best overall when compared to the other presented methods.

The paper is organized as follows. Section 2 presents the related work and further motivation for this research. Section 3 provides the background information relevant to this research and describes the assumptions behind our work. Section 4 describes the 3 proposed algorithms and section 5 discusses the experiments associated with this research. Section 6 presents the results of our experiments and section 7 provides concluding remarks and presents future work.

II. Related Work and Motivation

Previous research on direct communication cooperative learning agents focus on developing sharing strategies that are adopted by all agents in the environment. Tan [2] introduced cooperative learning methods where agents share sensations, episodic information, or learned policies to achieve improved learning rates. The work presented the concept of policy averaging (PA), in which agents average their learned policies together and exploit the other agents' shared knowledge to expedite learning. Another cooperative learning technique is to have agents make use of a joint policy table [3]. Using this method, several distinct agents explore an environment and update the same policy that is shared by all agents. Agents do not communicate directly with one another but rather to a central entity that stores the shared policy. In our work, we refer to this method as the Centralized sharing method and use it as an upper bound to compare our agents' learning performance against. A similar concept was proposed by Yang et al. with a blackboard-based communication method in which agent communication is relayed through and controlled by this central blackboard architecture [4]. Ribeiro et al. propose a cooperation model that uses a global action policy to ensure proper convergence. The global action policy unifies agents' partial action policies to produce optimal policies for the generic interaction model case [5]. Multi-agent cooperative learning has been shown to improve learning performance for both the homogeneous and heterogeneous cases [6].

The weighted-strategy sharing (WSS) measures the expertness of the cooperating agent's policies and weighs their contributions according to this value. This method has been shown to be effective when agent experiences differ [7], [8]. The WSS cooperative learning method could be considered a viable strategy to use in a non-reciprocating environment. However, results show that for the case where all agents start in a similar initial location, the agents' learning performance does no better than when using independent learning. Another technique for determining an agents area of expertise uses several classifiers [9] to extract partial policies containing expertise. Other techniques furthering the identification and exploitation of agents areas of expertise have also been presented [10].

III. Background

In this section we will briefly describe the mechanics necessary to understand the basics of RL in addition to the action-selection (AS) method used in this paper.

A. Reinforcement Learning

RL is a bottom-up programming methodology that gives agents the ability to extract salient environmental cues online. RL uses Markov decision processes (MDP) to model how an agent interacts with its environment. An MDP is a 4-tuple taking the form $(S, A, P^a_{ss'}, R^a_{ss'})$ where S is the state space, A is the action set, P is the transition function where $P^a_{ss'}$ represents the probability of transitioning from state s to state s' via action a, and R is the reward function where $R^a_{ss'}$ represents the expected value of the reward achieved when an agent moves from state s to state s' via action a. As an agent explores its environment, it updates its policy function π that maps each state $s \in S$ and action $a \in A(s)$ to $\pi(s, a)$. The agent attempts to maximize the expected total sum of rewards gained over time to converge to an optimal policy π^* (multiple may exist).

For this paper, we use a popular form of RL called Q-Learning: a type of temporal-difference (TD) learning where $Q : S \times A \rightarrow \mathbb{R}$. TD learning is a learning technique in which an agent will update its previously estimated state values using the differences between its current and former values. This effectively propagates more accurate estimates of the state values as learning continues. Agents define an action-value function for policy π by $Q^\pi(s, a)$ where the agent takes action a in state s and then applies policy π. Q-Learning represents an off-policy form of TD control which, for the one-step case used in this paper, takes the following form:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t)\Big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big]. \quad (1)$$

where t is the time step parameter, α is the learning rate parameter, and γ is the discount rate parameter. In this paper, the single-agent form of RL is applied to each agent in a multi-agent environment.

B. Agent Action-Selection

As agents navigate the environment, various strategies for selecting an action in each state can be used, some of which take into account the exploration-exploitation trade-off. Typical strategies include random selection, optimal selection, ϵ–greedy selection, ϵ–decreasing selection, and softmax selection [1]. In this paper we use softmax AS. Agents use the softmax AS rule with the Boltzmann distribution, as seen in (2), to determine the probability of selecting action a_i in state s.

$$P(a_i|s) = \frac{\exp(Q(s,a_i)/\tau)}{\sum_{a \in A(s)} exp(Q(s,a_k)/\tau)}. \quad (2)$$

where τ represents the temperature parameter. As $\tau \to \infty$ the probability of selecting actions becomes more random. As $\tau \to 0$ the probability of selecting the actions with higher Q-values becomes more likely.

C. Key Assumptions

We make the following 3 assumptions in this work that persist throughout the paper:

1) Agents are assumed to be using Q-Learning. The agents have a public interface that allows other agents to query their Q-tables. This is a minimal method for catalyzing cooperative learning.
2) All agents have the same Q-Learning parameter values.
3) The agent(s) employing our method knows how many other agents are in the environment and has enough memory to store the Q-tables and E-tables for them.

IV. The Algorithms

In this section we describe the architecture of the agent in addition to the 3 algorithms we have developed.

A. General Agent Structure and Behavior

Traditional agents in a RL simulation have a single policy representing their memory. Our approach gives the agent the memory capacity to embed the policies of the other agents.

Definition 1: An agent's memory, M, is a set of N policies $\{\pi_0, ..., \pi_{N-1}\}$ where π_0 is the agent's own, or self, policy and policies $\{\pi_1, ..., \pi_{N-1}\}$ are shared policies. Each shared policy $\pi_i \in M$ is a policy from each of the other $(N-1)$ agents.

Our approach separates the two types of policies in memory and updates both accordingly. The agent updates its self policy when an action is directly taken by the agent and therefore experiences being in state s_t, taking action a_t, and arriving at state s_{t+1} with reward r_{t+1}. The agent updates its shared policies after each time step by querying the other agents for their Q-tables.

There are 5 steps taken by the agent at each time step: 1) Action Selection: Take a step with action a in state s using policy π_0 to make a decision, 2) Update π_0 based on the experience gained by taking action a in state s, 3) Query Q-values from the other agents' policies and update policies $\{\pi_1, ..., \pi_{N-1}\}$ accordingly , 4) Assess/evaluate the best values to use, and 5) Combine the values to create an updated policy, π_0 (that is then accessible to others). This interaction cycle is shown in Fig. 2. Our proposed methods operate at the level of steps 4 and 5 and are differentiated by the actions taken in these steps. Note that the 3 methods presented below still perform learning by trial-and-error, as in step 2. These methods simply supplement the agents' knowledge by exploiting the other agents' knowledge.

B. Algorithm 1: Modified Averaging (MA) Method

The PA method first suggested by [2] is a simple sharing strategy that averages all agent policies together – according to each (s,a) pair – and assigns the averaged policy to all agents. After all agents have taken their step for time step t in the episode, the PA method averages the Q-value of each action $a \in A(s) \forall s \in S$ with all other corresponding actions in their corresponding states across all agent policies. The method then overwrites the Q-values across all policies for each action $a \in A(s) \forall s \in S$ with the corresponding average for that action. As all agents participate and use the same averaged policy, this is considered a reciprocating sharing strategy.

We propose the MA method, a non-reciprocating version of the PA method. The MA method, as seen in (3), is similar to the PA method except that after averaging all agents' policies, only π_0 is assigned the averaged Q-table because the other agents are not participating in the sharing method.

$$\text{MA}(s,a) = \frac{1}{n} \sum_{i=0}^{n-1} Q^{\pi_i}(s,a) \quad (3)$$
$$\text{for } s \in S, a \in A(s), \pi_i \in M$$

C. Algorithm 2: Modified Experience Counting (MEC) Method

The experience counting (EC) method, also referred to as the non-trivial update counting method in [11], is a popular sharing strategy that has been shown to outperform the PA method. The EC method makes use of a E-table, or experience table, in addition to a Q-table to keep track of which states and what actions an agent has actually experienced during the simulation. The central idea behind this method is that the most experienced agent with regard to each (s,a) pair should have the most contribution to the Q-table that is synchronized by all agents after each step.

Each (s,a) visitation results in the corresponding (s,a) entry of the E-table being updated by increasing the visitation, or experience, count for that entry by 1. After all agents have taken their step for time step t in the episode, the EC method goes through each (s,a) pair in the Q-table and allows the agent with the most experience for that (s,a) pair to contribute its current Q-value for (s,a) to

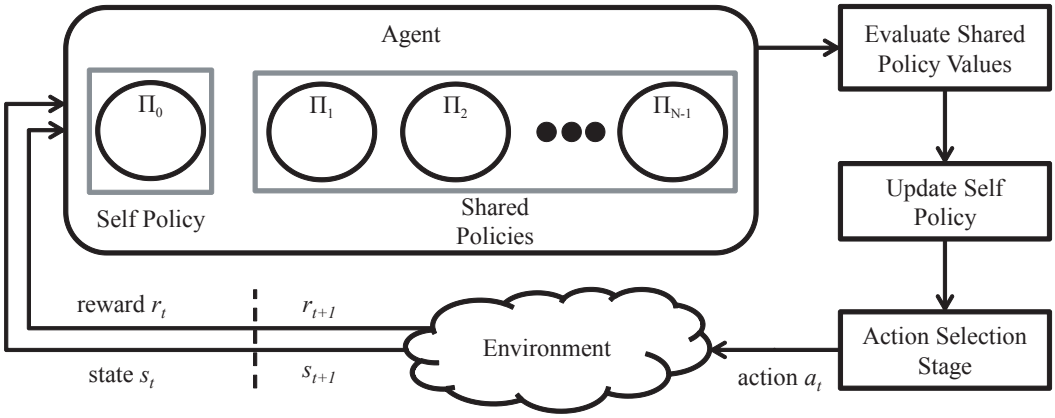

Figure 2. Extended RL agent-environment interaction cycle.

the resultant Q-table that will overwrite all agent policies. Experience ties are resolved by randomly selecting from the tied experience values, which correspond to Q-values. Similarly, each (s,a) entry in the E-table for all agents is updated to the highest count experienced out of all agents because the corresponding entry in the Q-table was updated. At the end of each step all agents have the same Q-table and E-table.

We propose the MEC method, as seen in (4), a non-reciprocating version of the EC method. Because an agent employing the MEC method cannot assume the other agents are reciprocating, it cannot rely on receiving others' E-tables for counting experience. Instead, we propose that the agent queries and stores the other agents' Q-tables in M. Additionally, the agent will store a E-table for each of the other agents in M as well. The agent can then observe which Q-values have changed since the last time step t and will associate this with an experience point for that particular (s,a) entry in the E-table for that agent. Therefore, the essence of the EC method is captured and the agent can reconstruct the (s,a) pair(s) that the other agents have experienced to determine which agents should contribute to the policy that will be assigned to π_0. We propose 3 variations of the MEC method for testing: Any, Positive, and Negative. The MEC – Any variation assigns an experience point when a Q-value for (s,a) at time step t does not equal the Q-value for (s,a) at time step $t+1$. The MEC – Positive variation assigns an experience point when a Q-value for (s,a) at time step t is less than the Q-value for (s,a) at time step $t+1$. Finally, the MEC - Negative variation assigns an experience point when a Q-value for (s,a) at time step t is greater than the Q-value for (s,a) at time step $t+1$.

$$\begin{gathered} \text{MEC}(s,a) = Q^{\pi_j}(s,a) \\ \text{where } j = \arg\max_i E^{\pi_i}(s,a) \\ \text{for } s \in S, a \in A(s), \pi_i \in M \end{gathered} \tag{4}$$

D. Algorithm 3: Hybrid Experience Counting and Averaging (HECA) Method

The HECA method is a hybrid method combining elements from both the MA and MEC methods. This method is similar to the MEC method except that experience, or E-table, ties are resolved by averaging the Q-values associated with the tied experience values. We propose only 1 variation of the HECA method for testing: Any. The Any variation assigns an experience point when a Q-value for (s,a) at time step t does not equal the Q-value for (s,a) at time step $t+1$.

V. Experiments

A. Maze World Domain

To explore the methods we propose, an 8×8 grid-based environment is used to simulate the maze world domain [12]. Agents attempt to navigate to the goal location with the intention of reaching it using the fewest number of steps. The set of states S is composed of each cell position (x,y) on the map. $|S| = c \times d$ where c is the number of cells in the horizontal direction and d is the number of cells in the vertical direction. In each state, agents can move one step and choose from the following action set: A={North, East, South, West}. Agents that attempt to move into a wall are blocked and will remain in the same position. Similarly, if an agent attempts to leave the bounds of the map, the agent will remain in the same position. Agents are not aware of the location of the other agents and may occupy the same cell. Agents take turns taking steps. For time step t agent 1 takes a turn, followed by agent 2, and so forth until all agents have taken a turn for time step t. Agents start each episode in the cell with position (0,3). The goal cell is located in position (7,3). For every step an agent takes that leads to a non-goal state the agent receives a reward of -1.5. Conversely, actions that lead the agent into a goal state provide the agent with a reward of 20. The maze world map used in testing was randomly generated with respect to wall placement. See Fig. 3 for an example of the maze world environment.

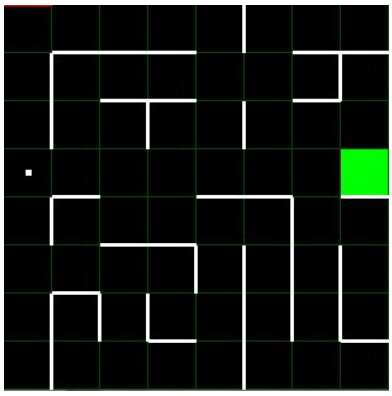

Figure 3. An 8 × 8 maze world domain map. The white lines are walls, the white dot is an agent, and the green cell is the goal.

B. Evaluation

An episode consists of the steps taken after an agent starts in the initial state and spans until the agent reaches the goal state. A run is defined by a series of episodes. The number of episodes per run is determined by the number of episodes that are necessary to allow the learning performance of the agents to stabilize. Agent policies are carried over from one episode to the next, but not from run to run. In our experiments, we evaluate the learning performance of the agents over 500 runs, where each run consists of 60 episodes. For the simulation parameters we set α = 0.15, γ = 0.99, and τ = 0.4; these are default parameters for the maze world domain [11].

For each of the 3 experiments in this paper, we present a graph comparing the average number of steps per episode for Agent 0, or A_0, vs. the number of cooperating agents in the simulation for the proposed sharing methods. The average number of steps per episode signifies learning performance. These graphs provide visual results displaying the trends associated with each of the sharing methods. We vary the number of agents up to 8 because, at this point, we near performance convergence for the sharing methods. For each experiment, we also include a table displaying the average number of steps per episode for A_0 for the case when the simulation has 8 agents. As aforementioned, the sharing method performance converges when the agent count is 8 and therefore these step averages are good overall indicators of the performance for the sharing methods. We also include the averages' respective 95% confidence intervals (CI) calculated by a t-test. A CI allows us to determine whether or not the differences in learning performance, as indicated by average steps per episode, are statistically significant.

To simulate testing environments with varying sharing strategies and to test the effectiveness of our proposed methods, we have set up 3 unique environments that represent a range of the types of sharing strategies that may be encountered. A_0 will employ one of the 3 proposed sharing methods while the other agents, $[A_1, ..., A_{N-1}]$ will employ one of the following methods: 1) Independent: No sharing strategy is employed and each agent's Q-table will likely be different from the other agents' Q-tables. At each time step t, at most 1 Q-value in the Q-table will change. 2) PA: The agents use the PA sharing method. Agents using the PA method will have the same policy π after each time step t because the method dictates that agents synchronize their Q-tables after averaging the values. Therefore, it is possible for multiple Q-values to change between time steps. 3) EC: The agents use the EC sharing method. Again, agents will have same policy π after each time step t and it is possible for multiple for multiple Q-values to change between time steps. For each of the experiments we compare against 4 reference methods: Centralized, PA, EC, and Independent. The Centralized sharing method represents an upper bound for learning performance as all agents write to the same policy after each step. Conversely, the Independent method serves as a lower bound for performance as any methods that perform worse than it might as well learn independently. The 4 reference methods are run in an environment where all agents are using that specific sharing method. Inclusion of the PA and EC reference methods allows us to compare the performances of the reciprocating and non-reciprocating versions of the methods.

VI. Results

A. Experiment 1

In this experiment, we explore how the proposed algorithms perform for A_0 when agents $[A_1, ..., A_{N-1}]$ are using the Independent sharing method. Results are shown in Fig. 4. The results indicate that as the number of agents increases, the average number of steps per episode for each method decreases. Intuitively, this makes sense because the agents are able to take advantage of more knowledge. Once the sharing method's performance stabilizes, it is clear that the EC and Centralized methods perform best and the Independent method performs worst. Table I shows that, according to t-tests, the differences in performance among all listed methods are statistically significant. The MA method performs 1.12 times more poorly than the PA method when there are 8 agents in the environment. This is expected as the PA method operates with all agents participating in synchronizing Q-tables. However, the MA method still outperforms Independent learning. This indicates that it is possible for agents to improve learning performance in environments where other agents do not reciprocate. According to the results, the MEC – Any method performs the best out of the proposed methods and only performs 2.44 times more poorly than the upper bound for learning performance. The HECA – Any method performed better than the MA method by a factor of 1.19 but performed worse than the MEC – Any method by a factor of 1.20. This experiment indicates that the MEC method performs

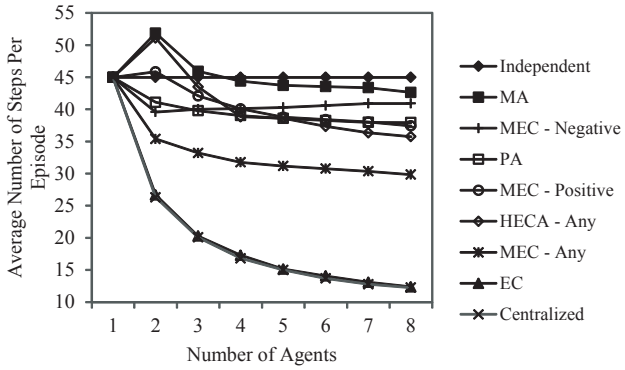

Figure 4. Simulation results for A_0 when the other agents use the independent sharing method

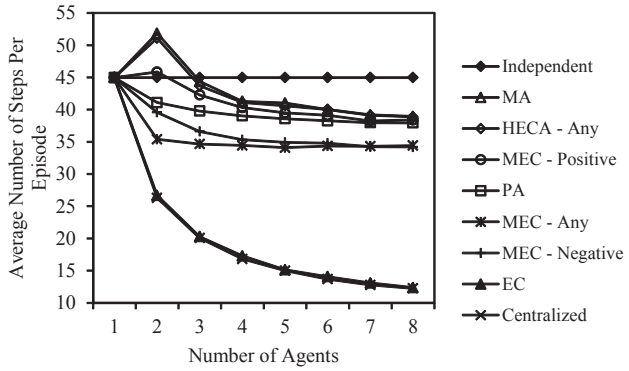

Figure 5. Simulation results for A_0 when the other agents use the PA sharing method

Table I
SIMULATION RESULTS FOR A_0 WHEN THE OTHER AGENTS USE THE INDEPENDENT SHARING METHOD

Sharing Method	Average Steps with 95% CI
Independent	44.98 ± (0.33)
MA	42.64 ± (0.37)
MEC - Negative	40.91 ± (0.30)
PA	37.94 ± (0.30)
MEC - Positive	37.42 ± (0.36)
HECA - Any	35.76 ± (0.33)
MEC - Any	29.84 ± (0.21)
EC	12.37 ± (0.11)
Centralized	12.24 ± (0.11)

Table II
SIMULATION RESULTS FOR A_0 WHEN THE OTHER AGENTS USE THE PA SHARING METHOD

Sharing Method	Average Steps with 95% CI
Independent	44.98 ± (0.33)
MA	38.94 ± (0.34)
HECA - Any	38.88 ± (0.35)
MEC - Positive	38.40 ± (0.35)
PA	37.94 ± (0.30)
MEC - Any	34.42 ± (0.26)
MEC - Negative	34.20 ± (0.27)
EC	12.37 ± (0.11)
Centralized	12.24 ± (0.11)

Table III
SIMULATION RESULTS FOR A_0 WHEN THE OTHER AGENTS USE THE EC SHARING METHOD

Sharing Method	Average Steps with 95% CI
Independent	44.98 ± (0.33)
PA	37.94 ± (0.30)
MEC - Negative	15.09 ± (0.18)
MEC - Positive	14.43 ± (0.19)
MA	13.99 ± (0.17)
MEC - Any	13.17 ± (0.13)
HECA - Any	12.67 ± (0.12)
EC	12.37 ± (0.11)
Centralized	12.24 ± (0.11)

well when the other agent's policies have different values and do not synchronize to the same policy after each time step t.

B. Experiment 2

In this experiment, we explore how the proposed algorithms perform for A_0 when agents $[A_1, ..., A_{N-1}]$ are using the PA sharing method. Results are shown in Fig. 5. Table II shows that, according to t-tests, the differences in performance among the MA and HECA – Any methods are not statistically significant. Similarly, Table II also shows that the differences in performance among the MEC – Any and MEC – Negative are not statistically significant. The MEC – Negative method outperforms the Independent learning method by a factor of 1.32 but performed worse than the upper bound by a factor of 2.79. For this experiment, the MEC method performs well when the other agent's policies have similar values and synchronize to the same policy after each time step t. Both experiment 1 and 2 confirm that the proposed methods can suitably adapt to both types of situations encountered for sharing strategies.

C. Experiment 3

In this experiment, we explore how the proposed algorithms perform for A_0 when agents $[A_1, ..., A_{N-1}]$ are using the EC sharing method. Results are shown in Fig. 6. Table III shows that, according to t-tests, the differences in performance among all listed methods are statistically significant. The HECA – Any method performs the best, outperforming the Independent method by a factor of 3.55 and performing slightly worse than the upper bound by a factor of 1.04. All methods perform significantly better in this experiment when compared to experiments 1 and 2 because the EC sharing method being used by agents $[A_1, ..., A_{N-1}]$ is an effective sharing strategy. Both the HECA – Any and MEC – Any methods perform well when the other agent's policies have similar values and synchronize to the same policy after each time step t.

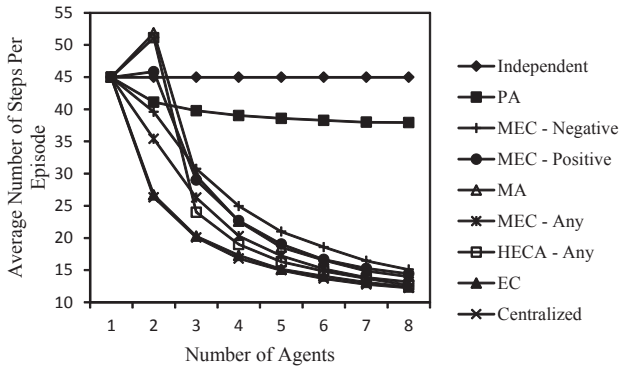

Figure 6. Simulation results for A_0 when the other agents use the EC sharing method

VII. Conclusions and Future Work

Results from the 3 experiments show that it is possible for agents to improve learning performance in environments where other agents do not reciprocate. The 3 proposed methods – MA, MEC, and HECA – all perform better than the Independent learning method. This indicates that by exploiting the pre-existing agent interface, learning performance can be expedited. Most notably, the MEC – Any method performed best overall across the 3 experiments.

This research opens up the possibility for future work. Currently, our method assumes that agents can communicate with and query the Q-tables of all agents in the environment, regardless of physical distance between the agents. In real situations where agents are robots and take on a physical form, this assumption may not hold. Future work could study how the methods proposed in this paper are affected by a limited communication range and propose modifications for these methods that perform better in such conditions. In this work we also assume that agents employing the proposed methods have enough memory to store the Q-tables and E-tables for all other agents in the environment. Again, if the agents were actual robots, this assumption of a large memory may prove to be unrealistic. Future work could study how the methods proposed in this paper are affected by a limited memory size and propose modifications for these methods that would enable the methods to perform in such environments. An implicit assumption with this work was also that agents knew how many other agents were in the environment in addition to assuming a reliable communication channel among the agents where Q-tables queries were always satisfied. The information bandwidths associated with our methods could be taken into account. Our method assumes unlimited bandwidth; however, for some RL applications that are used with physical agents, bandwidth is a limiting factor.

References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[2] M. Tan, "Multi-agent reinforcement learning: Independent versus cooperative agents," in *ICML*. Morgan Kaufmann, 1993, pp. 330–337.

[3] H. R. Berenji and D. Vengerov, "Advantages of cooperation between reinforcement learning agents in difficult stochastic problems," in *In Proceedings of 9th IEEE International Conference on Fuzzy Systems*, 2000.

[4] Y. Yang, Y. Tian, and H. Mei, "Cooperative q learning based on blackboard architecture," in *Proceedings of the 2007 International Conference on Computational Intelligence and Security Workshops*, ser. CISW '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 224–227. [Online]. Available: http://dx.doi.org/10.1109/CIS.Workshops.2007.36

[5] R. Ribeiro, A. P. Borges, and F. Enembreck, "Interaction models for multiagent reinforcement learning," in *CIMCA/IAWTIC/ISE*, M. Mohammadian, Ed. IEEE Computer Society, 2008, pp. 464–469.

[6] P. Zhang, X. Ma, Z. Pan, X. Li, and K. Xie, "Multi-agent cooperative reinforcement learning in 3d virtual world," in *ICSI (1)*, ser. Lecture Notes in Computer Science, Y. Tan, Y. Shi, and K. C. Tan, Eds., vol. 6145. Springer, 2010, pp. 731–739.

[7] M. N. Ahmadabadi, M. Asadpour, and E. Nakano, "Cooperative q-learning: the knowledge sharing issue," *Advanced Robotics*, vol. 15, no. 8, pp. 815–832, 2001.

[8] M. N. Ahmadabadi and M. Asadpour, "Expertness based cooperative q-learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 32, no. 1, pp. 66–76, 2002.

[9] M. N. Ahmadabadi, A. Imanipour, B. N. Araabi, M. Asadpour, and R. Siegwart, "Knowledge-based extraction of area of expertise for cooperation in learning," in *IROS*. IEEE, 2006, pp. 3700–3705.

[10] B. N. Araabi, S. Mastoureshgh, and M. N. Ahmadabadi, "A study on expertise of agents and its effects on cooperative q-learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 37, no. 2, pp. 398–409, 2007.

[11] L. Torrey and M. E. Taylor, "Help an agent out: Student/teacher learning in sequential decision tasks," in *AAMAS*. (In Press), 2012.

[12] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.