Non-Reciprocating Sharing Methods in Cooperative Q-Learning Environments

Bryan L. Cunningham

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Yong Cao, Chair
Yang Cao
Andrea L. Kavanaugh

August 9, 2012
Blacksburg, Virginia

Keywords: Cooperative Learning, Multi-Agent Reinforcement Learning, Information Exchanges in Multi-Agent Systems, Agent Interaction Protocols

Non-Reciprocating Sharing Methods in Cooperative Q-Learning Environments

Bryan L. Cunningham

(ABSTRACT)

Past research on multi-agent simulation with cooperative reinforcement learning (RL) for homogeneous agents focuses on developing sharing strategies that are adopted and used by all agents in the environment. These sharing strategies are considered to be reciprocating because all participating agents have a predefined agreement regarding what type of information is shared, when it is shared, and how the participating agent's policies are subsequently updated. The sharing strategies are specifically designed around manipulating this shared information to improve learning performance. This thesis targets situations where the assumption of a single sharing strategy that is employed by all agents is not valid. This work seeks 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, several intra-agent methods are proposed that do not assume a reciprocating sharing relationship and leverage the pre-existing agent interface associated with Q-Learning to expedite learning. The other agents' functions and their sharing strategies are unknown and inaccessible from the point of view of the agent(s) using the proposed methods. The proposed methods are evaluated on physically embodied agents in the multi-agent cooperative robotics field learning a navigation task via simulation. The experiments conducted focus on the effects of the following factors on the performance of the proposed non-reciprocating methods: scaling the number of agents in the environment, limiting the communication range of the agents, and scaling the size of the environment.

Acknowledgments

I am grateful for the opportunity to pursue my master's degree in Computer Science and owe many thanks to my adviser, Dr. Yong Cao, for his continued guidance and patience throughout this work. His support and dedication made this work possible.

I give my deepest appreciation to my family, for providing love and encouragement throughout my life. Many thanks to my sister, Elizabeth Zager, J.D. for proofreading my conference papers. Additionally, I would like to thank my beautiful fiancée and best friend, Devon Reed, for keeping me sane during the entire process.

I would like to thank my friends for some insightful discussions: Seung In Park, Felipe Bacim, Bireswar Laha, Stacy Branham, Bing Fang, and Jackie Pennysworth. Finally, I would like to thank the staff of the Computer Science department for their help and instruction over the past 6 years at Virginia Tech.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

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 [47]. 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 [50]. 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, when it is shared, and how the participating agent's policies are subsequently updated. 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 within the same environment, as shown in Fig. 1.1. We address how agents with no predetermined sharing partners can exploit groups of cooperatively learning agents to improve learning performance when compared to Independent learning.

Inspiration for our research stems from the work of Zamstein on a robot named Koolio [59]. Koolio is an autonomous delivery robot, capable of learning to navigate hallways to reach specific destinations. Koolio makes use of Q-learning and uses sensor input and a rewards scheme to successfully accomplish its tasks. The Koolio research project is one of many similar projects involving a physically embodied agent in a real-world environment making use of RL. It is quite easy to envision applications where there may be a need for multiple robots in the environment capable of independently carrying out individual tasks – while sharing learned knowledge with other agents to expedite the learning process. For example, some hospitals are introducing robots into the environment for trivial tasks such as medication delivery [36]. These robots serve as a means to automate tasks to improve efficiency.

One can imagine that as more robots are introduced into a hospital, distinct groups of cooperating agents begin to develop as they may consist of differing capabilities and internal structures. These differences in design may necessitate a variety of cooperative sharing algorithms among the groups. This is a problem from a learning efficiency standpoint, as reciprocating agents will look only to their predetermined sharing partners with the same sharing method for additional knowledge. Agents employing non-reciprocating methods will be able to overcome this learning inefficiency and make use of the knowledge accumulated by the other agents, regardless of their employed sharing method.

Additionally, consider the following cases. What if a new agent is introduced into the environment that does not know the sharing strategy of the other agents, but it still wants to exploit their knowledge. Alternatively, perhaps the agent cannot use the same sharing strategy due to hardware or computational restrictions. Finally, consider the case where there are 2 groups of agents in the environment that are utilizing their own sharing methods but each want to exploit the knowledge of the other group. The concept of non-reciprocating sharing methods provides a solution to these problems by making agent–group and group–group collaboration possible. To the best of our knowledge, research into non-reciprocating methods has not been addressed within the cooperative Q-learning field for homogeneous agents.

Another relevant application scenario for this work includes disaster site work. For example, robots of differing capabilities utilizing a variety of cooperative learning sharing methods may be deployed by different agencies for search and rescue missions. It would be beneficial to allow the disparate groups of agencies deployed by the agencies to be able to share the knowledge gained in the environment in order to improve learning performance.

Note that in this work we focus on agents with varying memory capabilities and sharing methods. Extensions of this work could investigate the potential for non-reciprocating methods in environments with agents of other varying internal capabilities, such as differing sensors, actions, etc, but such work is not covered in this thesis.

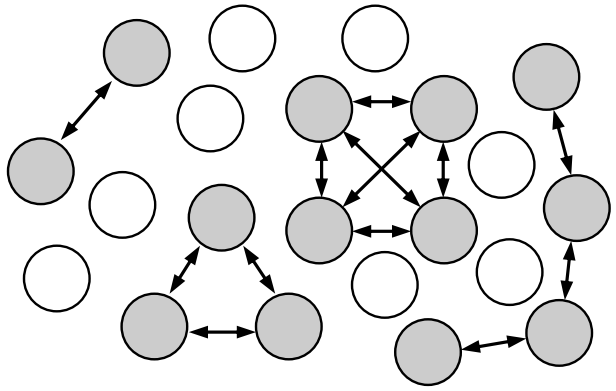


Figure 1.1: Separate groups of cooperative learning agents employing distinct sharing strategies interspersed with individual learning agents.

1.2 Research Goals

The main goal of this research is to explore and understand the potential use for non-reciprocating sharing methods in homogeneous, cooperative multi-agent RL environments. Specifically, we seek to explore the following questions:

1) How do the proposed methods perform in environments where other agents do not reciprocate using the same sharing method? Given a memory representation table, or Q-table, from another homogeneous, cooperative agent, what is the best method to use to extract salient data to improve learning performance?

2) How does the learning performance of the proposed methods compare to the learning performance of the (reciprocating) reference methods?

3) How does scaling the number of agents in the environment affect the learning performance of the non-reciprocating methods?

4) How does limiting the communication ranges of the agents affect the learning performance of the non-reciprocating methods? Is there any way to improve the learning performance of the methods in such conditions?

5) How does scaling the environment size affect the learning performance of the proposed methods?

Note that there are several possible metrics by which to evaluate the effectiveness of our proposed approaches: learning performance, bandwidth consumed, memory usage, and algorithmic complexity, to name a few. This problem can be approached from many different perspectives in terms of designing the algorithms to improve upon a specific metric. In

this work, we focus on the learning performance metric with our goal being to design algorithms that attain optimal learning rates using few training trials – this is our work's primary objective. We seek to design algorithms in which agents 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. Essentially, we are striving to design algorithms in which agents maximally and beneficially assimilate others' collective, available knowledge for the purpose of expediting learning performance. The agents must be able to simultaneously further develop their own policies in addition to exploiting others' policies. The algorithms also must assess/evaluate the policies of the other agents effectively and be capable of combining the knowledge stored in agent policies to achieve improved learning performance rates. Note that we do consider the other 3 metrics, however, they are not our primary focus.

1.3 Approach

Several intra-agent methods are proposed that do not assume a reciprocating sharing relationship and leverage the pre-existing agent interface associated with Q-Learning to expedite learning. The other agents' functions and their sharing strategies are unknown and inaccessible from the point of view of the agent(s) using the proposed methods. The proposed methods are evaluated on physically embodied agents in the multi-agent cooperative robotics field learning a navigation task via simulation.

The experiments we run will demonstrate if it is possible for agents to improve learning performance, when compared to the Independent method, in environments where other agents do not reciprocate with it. They will also indicate if, by exploiting the pre-existing agent interface, learning performance can be expedited even when exposed to a variety of environmental constraints.

Our work provides the following contributions:

1) 6 non-reciprocating methods that outperform the Independent learning method in environments where the other agents do not employ the same sharing method. This includes an alternative, improved method for calculating agent experience for state-action pairs by using a cumulative moving average across agent memory updates.

2) Detailed analysis on the effects of scaling the number of agents in the environment, limiting the communication range of the agents, and scaling the size of the environment for non-reciprocating methods.

3) Simulator with a GUI to create, modify, and visualize the environment and agents for a navigation task via Q-Learning.

4) Analysis tool to extract meaningful results from the experimental output associated with the simulator.

1.4 Outline

The remainder of this thesis is organized as follows. In Chapter 2, we summarize the main concepts associated with this research, like RL, MARL, etc., to provide the reader with a brief background of the material. In Chapter 3, we describe related work and elaborate on the necessity for this research. In Chapter 4, we discuss the details of the simulation, including the learning task, environment setup, evaluation criteria, and reference methods we compare our proposed methods against. In Chapter 5, we present the proposed non-reciprocating methods and the follow-up experiments we conduct on them. Additionally, we discuss an alternative agent structure in an attempt to improve upon the proposed methods. In Chapter 6, we present the limited communication range constraint tests and an improved non-reciprocating method to handle limited communication environments. In Chapter 7, we investigate the effects of scaling the environment size on the non-reciprocating methods and conduct a large-scale hospital simulation for visualization purposes. Chapter 8 provides a comparison of the top-performing proposed methods with regard to learning performance, communication bandwidth consumption, memory usage, and algorithmic complexity. Finally, in Chapter 9 we provide concluding remarks and present several topics that serve as extensions for future work. For convenience and sanity purposes, a list of the acronyms used in this thesis and their associated meanings are listed in the appendix.

Chapter 2

Background Knowledge

2.1 Reinforcement Learning (RL)

> "Imagine playing a new game whose rules you don't know; after a hundred or so moves, your opponent announces, 'You lose'. This is reinforcement learning in a nutshell." [42]

Russell and Norvig capture the essence of RL well in the previous quote. RL allows an agent to become immersed in an environment and learn, through feedback in the form of reinforcement signals or rewards, to accomplish a task. Note that the term 'agent' is ill-defined in the AI community, however, Panait and Luke have put forth a comprehensive definition that is often referenced. They define an agent as "... a computational mechanism that exhibits a high degree of autonomy, performing actions in its environment based on information (sensors, feedback) received from the environment" [38]. An agent is placed in an environment and must learn to perform successfully, where success is defined by criteria that can be specified by the designer via reward schemes; the agent attempts to maximize the total expected reward. The agent does not know how the environment operates nor does it know the effect of its actions.

The benefits of using RL to train an agent to learn a task are numerous, the most prominent being that it allows the human to avoid meticulously hard-coding the rules necessary for an agent to successfully accomplish its given task. Additionally, if examples were being used to teach an agent a task, as is the case with supervised learning, a large number of states and their appropriate classifications would be necessary to adequately train the agent. RL avoids both of these problems. For example, Ng et al. apply RL successfully towards autonomous helicopter flight, enabling the helicopter agent to perform many challenging maneuvers simply by specifying an appropriate rewards scheme [37]. Rule-based or example-based specification for the task is sometimes too complex to use for such a task. It is often hard to specify precisely what an agent should do in certain situations. However, allowing agents to explore the environment themselves, while providing rewards for specific states

when they are reached – successfully landing, deviating off course, crashing – can have extremely successful results.

This introduces a fundamental discussion in AI; top-down and bottom-up programming methodologies have emerged as the central dividing ideologies when it comes to categorizing artificial intelligence (AI) approaches [42]. The difference between the two being that the top-down approach explicitly programs the logic and reasoning ability necessary for autonomous thinking into the agent, whereas the bottom-up approach uses an elementally primitive canvas that requires the agent, or system, to explore and make the connections necessary to understand and utilize the given data. Reinforcement learning (RL) takes the bottom-up approach and simply provides the basic learning elements, or learning modules, necessary for an agent to be able to learn and then allows them to explore the environment, generating knowledge about the best action(s) to take in each state based on feedback given to them in the form of rewards. Unlike in the top-down methodology, the programmer does not have to envision and program each possible behavioral combination in order to achieve the desired results.

RL is a bottom-up programming methodology that imbues agents with the ability to generalize learned information and extract salient environmental cues online. RL relies on the concept of Markov decision processes (MDP) to model how an agent moves around in the 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 $\pi(s, a)$ which represents the probability of taking action a in state s. The agent attempts to maximize the expected total sum of rewards gained over time to converge to an optimal policy π^* (multiple may exist). Optimal policies share the same optimal action-value function Q* where

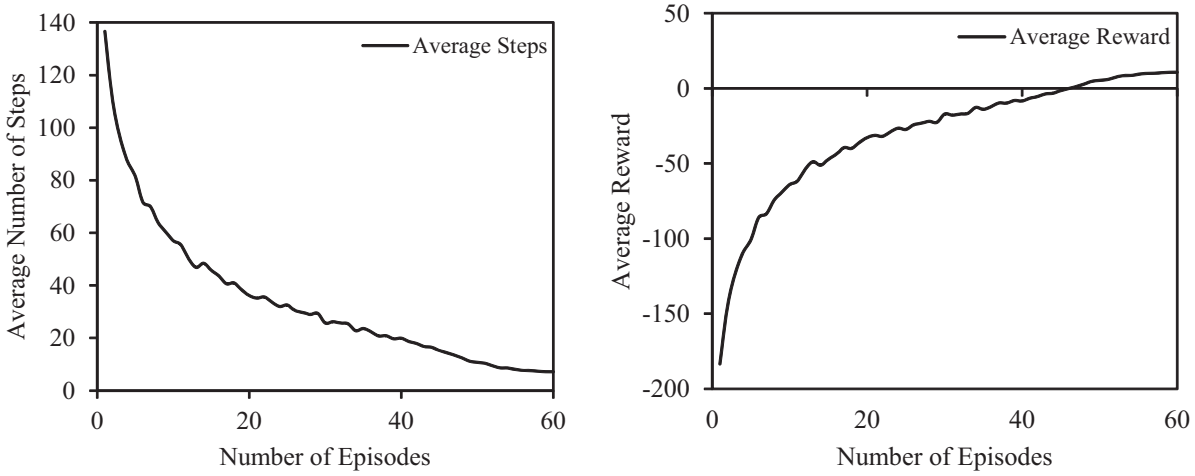
$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \forall s \in S, a \in A \tag{2.1}$$

Optimal policies must satisfy the Bellman optimality equation for Q*:

$$Q^*(s, a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right] \tag{2.2}$$

where $0 \leq \gamma \leq 1$ and is the discount rate. Simply put, Equation 2.2 states that the value of state s using the optimal policy equals the expected return of the best action a^* for s [47].

To provide a brief example of the results of running RL on a task, we introduce the learning task used in this paper: maze world domain. The goal of an agent in the maze world domain is to attempt to reach the goal position in as few steps as possible. Sample results for a



(a) Average Number of Steps Vs. Number of Episodes.

(b) Average Reward Vs. Number of Episodes.

Figure 2.1: Reward and Learning Performance Relationship.

simulation with 60 episodes are shown in Figs. 2.1a and 2.1b. Initially, the step count is high as the agent does a large amount of exploration. Over time, the agent learns more efficient paths to take and its step count converges to the optimal path. Notice that the reward graph in 2.1b shows inverse behavior where more episode experience results in higher accumulated reward values.

2.2 Multi-Agent RL (MARL)

Multi-Agent RL (MARL) is the extension of RL for multiple agents that share a common environment. MARL be accomplished in one of two ways: 1) Independent learners (ILs) where the single-agent form of RL is applied to each agent in a multi-agent environment, or 2) Joint action learners (JALs) where agents learn to make decisions in conjunction with the other agents through action coordination [14]. Agents using the IL approach lack the action coordinate skills and essentially treat the other agents as independent, dynamic forces acting within the same environment. Theoretical guarantees of convergence to optimality are no longer present due to the nonstationarity of the problem, however, [12] [52] point out that the use of ILs in MARL can be very successful and agents may still reach this optimal convergence point. We chose to use the IL approach for this paper as it is the most straightforward and allows for greater scalability, a key feature when considering environments that may contain many agents.

In the simulation we use for the experiments in this work, we focus on the communication aspect of this research and therefore do not allow agents to physically interact with one

another. Rather, agents can visually sense when other agents are near and may occupy the same physical space. Therefore, the learning problem is stationary.

2.3 Q-Learning

For this paper, we use a simple and popular form of RL called Q-learning: a type of temporal-difference (TD) learning where $Q : S \times A \rightarrow \mathbb{R}$ [47]. Basic pseudocode for the method is shown in Algorithm 1. 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 $Q^{\pi}(s, a)$ which indicates the expected return given that 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:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t) \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2.3}$$

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.

Algorithm 1 Q-Learning Pseudocode

```
Initialize Q(s, a) to 0
repeat
    Initialize s
    repeat
        Choose a from s using Q with Boltzmann action-selection
        Take action a, observe r and s'
        Q(s, a) ← Q(s, a) + α [r + γ max_a' Q(s', a') − Q(s, a)]
        s ← s'
    until s is terminal
until All episodes are complete
```

2.4 Agent Exploration/Exploitation and the Action-Selection (AS) Mechanism

Q-Learning dictates that agents visit each state-action (s,a) pair an infinite number of times to get an accurate Q-value for the pair. Since this is not possible to do, agents must use

another strategy to explore the state-action space. Agents can randomly explore the state-action space, however, this would take too long to obtain decent results. Another alternative for agents is to purely exploit knowledge gained from the environment. The downside to this approach is that agents often get stuck in local minima. Therefore, a popular solution to the problem is to have agents balance the task of exploration and exploitation at each step.

As agents navigate the environment, various control 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 [47]. In this paper we use softmax AS. Agents use the softmax AS rule with the Boltzmann distribution, as seen in Equation (2.4), to determine the probability of selecting action a_i in state s.

$$P(a_i|s) = \frac{e^{Q(s,a_i)/\tau}}{\sum_{a \in A(s)} e^{Q(s,a_k)/\tau}}. \tag{2.4}$$

where τ represents the temperature parameter and $\sum_a P(a|s) = 1$. 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. Because the equation uses e, $e^{Q(s,a_i)/\tau} > 0$ and therefore works for both positive and negative Q-values.

Chapter 3

Related Work

In this chapter we begin our discussion of related work by broadly defining the fields associated with our work. Following this, we narrow the scope of the related work and discuss recent research that inspired this work. In doing so, we provide reasoning for the necessity of this type of research and how our work explores a previously unstudied research void that runs tangential to current research in the field.

3.1 Multi-Agent Systems (MAS)

The broad field that encompasses our work is called cooperative multi-agent systems (MAS). Due to the complex nature of multi-agent systems, machine learning techniques have been applied to automate solutions to some of the problems. This field of automated MAS is called cooperative multi-agent learning [38]. A large amount of the research devoted to cooperative multi-agent learning studies systems that use RL, the focus of our work [12].

In this work we do not focus on open problems associated with MARL like scalability [1][56][20], problem decomposition [48][49][45], or credit assignment [7][25][51], instead, we are interested in the inter-agent communication associated with non-reciprocating agents. The MAS field can be broken down into many differing taxonomies. In this chapter, we provide a taxonomy that draws from several popular survey papers on the field [38][12][46]. Stone and Veloso state that the two major distinctions within MAS are homogeneous vs. heterogeneous agents and communicating vs. non-communicating agents. The authors reason that homogeneous agents have "the same internal structure including goals, domain knowledge, and possible actions" whereas heterogeneous agents may have different characteristics when compared to the other agents. In this paper, we consider homogeneous agents.

Communication is defined as agents exchanging some form of information with 1 or more other agents [46]. Within the heterogeneous field, agents can be either fully cooperative

[29][24][39], fully competitive[13][11], or a combination of both [23][10][26]. Inter-agent communication can be applied towards RL domains like maze world [15], predator-prey pursuit [50][19], soccer [2][43][8], and cooperative navigation [16], to name a few. Recently, studying and recreating crowd dynamics via RL has become a popular topic [52][17][32][31].

3.2 Communicating Agents

Communicating agents can be further broken down into 2 subcategories: indirect and direct communication.

3.2.1 Indirect Communication

Agents employing indirect communication utilize the "implicit transfer of information from agent to agent through modification of the world environment" [38]. This type of communication involves some form of environmental cue. The cue could be a physical object or marker left behind like pheromone trails [30][33][34][35] or another indicator like body position or pointing [54][40].

A popular form of indirect communication is imitation [28], where an agent will observe another agent in the same environment performing a task and attempt to use the same actions in the same, or similar, states to accomplish a task. Yamaguchi et al. developed 3 types of imitation: simple mimetism, conditional mimetism, and adaptive mimetism [57].

3.2.2 Direct Communication

Most cooperative learning research is accomplished by a specified 'teacher / pupil' training model, resulting in a unidirectional sharing relationship. The following work considers bidirectional relationships where all agents in the environment are capable of mutually helping other agents for the purpose of improving learning performance.

Previous research on direct communication cooperative RL for environments with 2 or more agents focuses on developing sharing strategies that are adopted by all agents in the environment. Tan [50] introduces cooperative learning methods where agents share sensations, episodic information, or learned policies to achieve improved learning rates. The work presents 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 allows agents to make use of a joint policy table [9]. 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 [58]. 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 [41]. Multi-agent cooperative learning has been shown to improve learning performance for both the homogeneous and heterogeneous cases [60].

Perhaps closest to our research is the work done by Ahmadabadi. 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 [4][3]. The WSS cooperative learning method could be considered a viable non-reciprocating strategy. 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 [5] to extract partial policies containing expertise. Other techniques furthering the identification and exploitation of agents areas of expertise have also been presented [6]. Our work in this thesis includes and expands upon our previous research on non-reciprocating sharing methods in cooperative Q-learning environments [18].

3.3 Extended AS Method Inspiration

The extended AS methods used in this paper were inspired by a technique called Probabilistic Policy Reuse (PPR). The PPR RL technique is a form of transfer learning in which an agent uses past learned (static) policies in conjunction with its own policy to improve learning performance [21][22]. The agent will gradually use the other policies less and rely on its own policy more as it strengthens over time. This approach is similar to ours in that agents make use of other policies; however, our approach makes use of actively developing policies. Our approach necessitates a different method to exploit the rapidly fluctuating policy values. The PPR technique judges which policy to exploit based on the policy as a whole, whereas we want to exploit policies based on state-specific measures.

More recent work extends this concept and uses the PPR method to allow agents to teach other agents how to learn through advice giving [53]. The work improves upon PPR to make the advice probabilities state-specific and nonuniform across all states. This is accomplished by using teacher confidence algorithms that bias the probability of using one policy over another for the student-teacher case.

While these papers contain similarities towards structuring the solution, our approach is different in that we do not attempt to wean the agents off of other policies. Instead of choosing one policy or another given by probability P to follow, we allow the agent to

choose an action from any of the policies with the probability defined across the entire set of actions by the Boltzmann distribution – effectively promoting better actions to be taken. Additionally, the PPR method is not designed for a cooperative learning environment.

Chapter 4

Simulation Details

In this section, we describe the details associated with our simulation. All code is written in C++ using the OpenGL graphics library. The code was executed on a Windows system. The GUI was directly inspired by Wharton's work [55].

4.1 Simulation Environment and Agents

The simulation environment chosen in this work to investigate the proposed non-reciprocating methods resembles that of the Koolio robot's environment. Robots utilizing RL that operate in real-world environments need to reduce the continuous nature of it into discretized chunks; one such method to do this is by using the tile coding technique [47]. Our grid-based environment essentially acts as an environment that has been reduced into fewer states using linear function approximation. As Q-learning is a tabular method, it is naturally suited to this type of discretized environment. This environment was also chosen because it allows for easy visualization of learned policies and results, is relatively computationally simple to test our proposed methods on, is easier to debug, and allows for quick map modification using the GUI we designed.

To explore the methods we propose, an (8×8) grid-based environment is used to simulate the maze world domain [27]. Agents take the form of physically embodied robots via simulation. The homogeneous 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 allowed to occupy the same cell in the environment - this

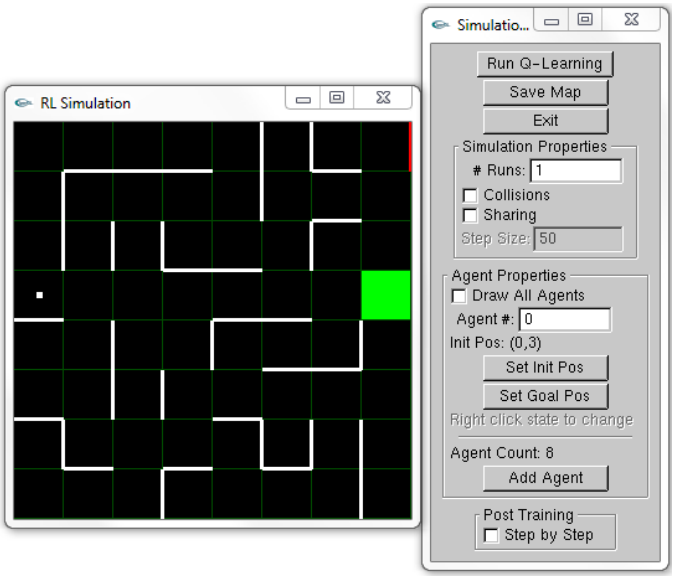


Figure 4.1: A randomly generated 8 × 8 maze world domain map. The white lines are walls, the white dot is an agent, and the green cell is the goal.

means that agents do not have to consider coordinating moves with other agents in addition to learning the navigational task, allowing us to easily extract the effects of cooperative learning on a simple task. 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. Given that most RL systems employ rewards that may be either positive or negative, we felt that testing in an environment with both a negative and a positive reward would be a more beneficial contribution to the research and would help uncover the methods that are more versatile. The maze world map used in testing was randomly generated with respect to wall placement. See Fig. 4.1 for an example of the maze world environment.

4.2 Running a Simulation

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 (with the exception of the 'Scaling Map Size' section, which is discussed later). Note that 500 runs are

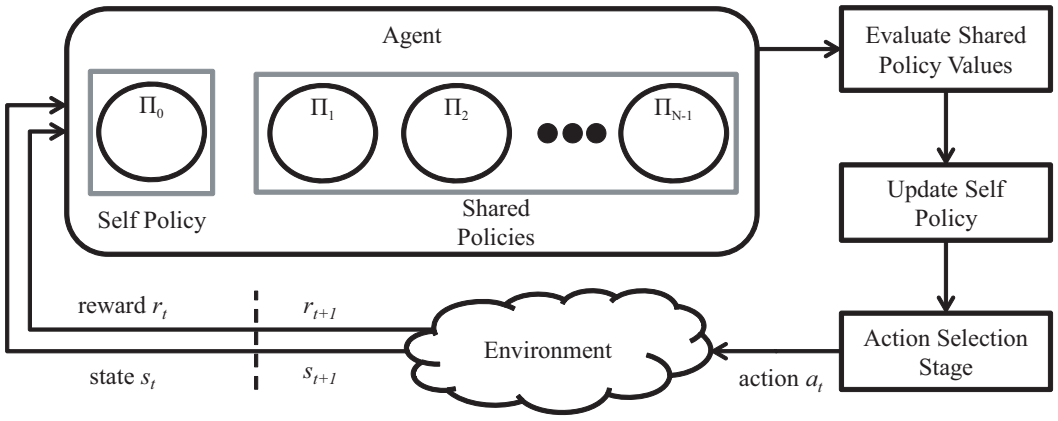


Figure 4.2: Extended RL agent-environment interaction cycle.

necessary to achieve results that provide consistent averages. For the simulation parameters we set $\alpha = 0.15$, $\gamma = 0.99$, and $\tau = 0.4$; these are default parameters for the maze world domain [53].

4.2.1 Simulation Flow

In general, agents go through 5 phases 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-tables from the other agents and update policies $\{\pi_1, ..., \pi_{N-1}\}$ accordingly
4) Assess/evaluate the best values to use
5) Utilize the values to create an updated policy, π_0 (that is then accessible to others)

This interaction cycle is shown in Fig. 4.2. Traditional single-agent RL only uses phases 1 and 2. Our proposed non-reciprocating methods go through all 5 phases. Our proposed methods simply supplement the agents' knowledge by exploiting the other agents' knowledge. Some methods perform additional work at each phase, as will be discussed later.

4.2.2 Simulation Evaluation

There are a number of different ways we evaluate the large amount of data collected in our experiments. For each of the experiments, we present a graph comparing the average

number of steps per episode for 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 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. Additionally, experiments are sometimes accompanied by a graph displaying the average number of steps vs. number of episodes. These step averages are calculated across all 500 runs for A_0 when there are 7 other agents in the environment simultaneously learning.

The difference between graphs with the average number of steps per episode and graphs with the average number of steps is made clear in Fig. 4.3. In Fig. 4.3, we show the results of a mock test that consists of 3 runs with 5 episodes per run. The average step count across each episode for all 3 runs is calculated in the overall results section (cells D:11 through D15). Additionally, an average of the averaged step counts is calculated in cell D:17; this is simply an average of all step count values across all episodes for all runs. The value in cell D:17 is what we use for the 'average number of steps per episode' value in our graphs. The values in cells D:11 through D:15 are what we use for the values in the average number of steps vs. number of episodes graphs.

4.3 Reference Methods

In our tests, we compare the results of our proposed methods 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 ideally serves as a lower bound for performance as any sharing methods that perform worse than this non-sharing method might as well learn independently. Inclusion of the PA and EC reference methods (discussed in more detail later) allows us to compare the performances of the reciprocating and non-reciprocating versions of the methods. The 4 reference methods are run in an environment where all agents are using that specific sharing method, i.e., the reciprocating versions.

	A	B	C	D	E
1	**Run 1:**		**Run 2:**		
2	Episode Number	Total Steps	Episode Number	Total Steps	
3	1	100	1	95	
4	2	90	2	85	
5	3	80	3	75	
6	4	70	4	65	
7	5	60	5	55	
8					
9	**Run 3:**		**Overall Results:**		
10	Episode Number	Total Steps	Episode Number	Total Steps	
11	1	90	1	95	= AVERAGE(B3,D3,B11)
12	2	80	2	85	= AVERAGE(B4,D4,B12)
13	3	70	3	75	= AVERAGE(B5,D5,B13)
14	4	60	4	65	= AVERAGE(B6,D6,B14)
15	5	50	5	55	= AVERAGE(B7,D7,B15)
16					
17	Average Number of Steps Per Episode:			75	= AVERAGE(D11:D15)

Figure 4.3: Results Spreadsheet Breakdown.

Chapter 5

Non-Reciprocating Methods

5.1 Non-Reciprocating Methods Approach

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. In this section we contribute 6 sharing methods: the modified averaging (MA) method, the modified experience counting (MEC) method, the hybrid experience counting and averaging (HECA) method, the pure self vs. shared (SVS) method, the average action value (AAV) method, and the highest action value (HAV) method. The first 2 methods adapt existing sharing strategies to perform in environments where other agents do not reciprocate. At the end of this section we analyze our methods by testing them in 3 unique environments where the other agents employ a variety of standard sharing methods with common learning characteristics.

In this section we describe the architecture of the agent, 3 key assumptions for our methods, and the methods we have developed to utilize the architecture. We divide the 6 presented methods in this section into 2 categories: regular and extended action-selection methods.

5.1.1 Agent Structure

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. Note that for this paper only Agent 0, or A_0, utilizes the agent structure described in this section. The other agents in the environment will retain the standard agent structure with 1 policy, π_0, for memory to store a Q-table and potentially an E-table for counting experience, depending on the sharing method used.

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 2 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.

5.1.2 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.

5.1.3 Regular AS Methods, Description and Design

The 3 methods presented in this section make use of the regular AS mechanism for allowing an agent to choose action a, given state s. This means that when the agent goes to select an action, it chooses from its own policy, π_0.

Modified Averaging (MA) Method

The PA method first suggested by [50] 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 Equation (5.1), 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.

$$\mathrm{MA}(s,a) = \frac{1}{n}\sum_{i=0}^{n-1} Q^{\pi_i}(s,a) \text{ for } s \in S, a \in A(s), \pi_i \in M \tag{5.1}$$

The MA method equally weights the contributing Q-values of the other agents and the agent employing the MA method. Therefore, the resulting Q-value for each (s,a) pair is a combination of the other Q-values for the (s,a) pair from each agent, with no preference towards Q-values that may be better than others. This method is useful for scenarios where other agents have Q-tables that are as good as or better than the agent employing the method. If not, incorporating and exploiting the knowledge of the other agents using this method may actually hinder learning performance. As is, this method provides no means for determining whether another agent's Q-table will be beneficial or not, so this method must be used in environments where some level of expectation regarding the quality of the other agent's policies can be ascertained beforehand.

Modified Experience Counting (MEC) Method

The experience counting (EC) method, also referred to as the non-trivial update counting method in [53], 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, as shown in Fig. 5.1. 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 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 Equation (5.2), a non-reciprocating version of the EC method. Because an agent employing the MEC method cannot assume the other

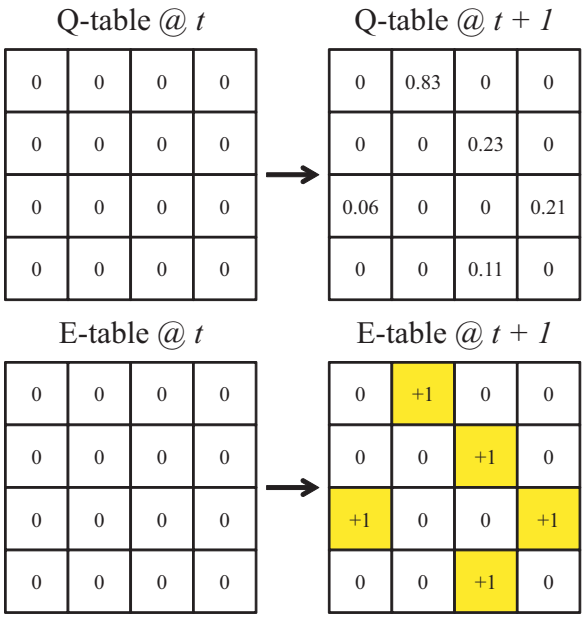


Figure 5.1: Updating the E-table.

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 create 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. The algorithm 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 $t+1$.

$$\begin{aligned} \text{MEC}(s,a) = & Q^{\pi_j}(s,a) \text{ where } j = \arg\max_i E^{\pi_i}(s,a) \\ & \text{for } s \in S, a \in A(s), \pi_i \in M \end{aligned} \tag{5.2}$$

The MEC method does not equally weight the contributing Q-values of the other agents and the agent employing the MEC method. Therefore, the resulting Q-value for each (s,a) pair is not a combination of the other Q-values for the (s,a) pair from each agent, with a strong preference towards Q-values that may potentially be better than others. This method is useful for scenarios where other agents have varying levels of expertise within their Q-tables. Another agent may have no experience with regard to (s,a) but have lots of experience with (s',a'). The MEC method would be able to successfully deduce that the (s,a) Q-value should not be used in the resultant Q-value but that the (s',a') Q-value should, since it

provides beneficial information. Whereas with the MA method, the agent would only want to use the method with agents that have Q-tables that are as good as or better than the agent employing the method, this MEC method would work well in environments where this is an uncertain characteristic. The MEC method has finer control over the (s,a) pairs used in resulting Q-table. Incorporating and exploiting the knowledge of the other agents using this method would never hinder learning performance, unless the other agents were somehow regressing in terms of learned knowledge. This method provides a built-in means for determining whether another agent's Q-table will be beneficial or not, and therefore this method may be used in environments where there is no level of expectation regarding the quality of the other agent's policies beforehand.

Hybrid Experience Counting and Averaging (HECA) Method

The HECA method is a hybrid method combining elements from both the MA and MEC methods. The algorithm 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 $t+1$. 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. All Q-values for (s,a) at time step t that have equal experience values should be given equal representation in determining the Q-value for the resultant Q-table. Instead of randomly selecting one of the Q-values associated with one of the tied E-table values, this method averages the Q-values associated with the tied E-table values for (s,a).

Like the MEC method, the HECA method does not equally weight the contributing Q-values of all the other agents and the agent employing the MEC method. Therefore, the resulting Q-value for each (s,a) pair is not a combination of all of the other Q-values for the (s,a) pair from each agent, with a strong preference towards Q-values that may potentially be better than others. The HECA method averages the Q-values associated with the tied E-table values for (s,a) instead of randomly selecting one of the Q-values associated with one of the tied E-table values because this enables equal input from the other agents for the resultant Q-values instead of choosing to incorporate the input of a single agent. In theory, this method should improve the learning performance for the agent employing the method over that of an agent employing the MEC method because it allows for further exploitation of the policies of the other agents.

5.1.4 Extended AS Methods, Description and Design

The 3 methods presented in this section make use of the extended AS mechanism for allowing an agent to choose action a, given state s.

Past research on multi-agent simulation with cooperative RL focuses on the development of various sharing strategies designed for the purpose of accelerating agent learning [4]. These

sharing strategies dictate how the agents should cooperate with each other to develop their learning policies more efficiently. The strategies are hard-coded to ensure the agents know what information to share, when to share it, and who to share it with to improve learning performance. These scripted approaches take time to develop and need to be carefully tuned to exploit policies effectively for all agents.

The following 3 methods take a different approach to sharing: allow the agents themselves to learn which parts of the other agents' learning policies to exploit for the purpose of accelerating their own learning. Specifically, this design enables agents to simultaneously learn the task at hand along with which agents and in what situations it should take advantage of cooperative learning to better its own learning performance. Using this approach, agents learn how to most effectively incorporate other agents' knowledge into their own policy to improve their own learning performance.

The 3 methods modify the standard RL method in 2 ways: increasing agent memory to include the policies of all cooperating agents and expanding the action-selection (AS) mechanism to include these new policies for making decisions. Using our method agents learn to balance exploitation of their own policies and those policies of the other agents while continuing to explore the environment through random exploratory actions. This design enables agents to use guidance from other currently developing policies to supplement their own individual knowledge. These cooperative learning methods represent a natural paradigm for agent learning and draws parallels to the way that humans incorporate others' information while learning.

This differs from the regular AS methods because the AS mechanism for those methods only takes $\pi_0 \in M$ into account whereas these methods will make an AS across $\{\pi_0, ..., \pi_{N-1}\} \in M$.

Pure Self vs. Shared (SVS) Method

When the agent goes to select an action, it takes into account the knowledge of the other agents. The expanded AS mechanism the agent uses acts across all policies in the agent's memory, M, by incorporating the probabilities of the actions from all policies into the decision-making process.

The pure SVS approach separates the two types of policies in memory and updates both accordingly. The agent updates its own 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 according to the function used for collecting policies from the other agents. For this method, the collection function is direct policy copy.

The Pure SVS method is more like the MEC method than the MA method with regard to determining the resultant Q-value for each (s,a) pair because a single Q-value from the

available Q-values is chosen to represent the resultant Q-value, rather than an averaged Q-value. This method differs from the presented regular AS methods because the Q-values for each action in the current state s are not evaluated and/or manipulated. Rather, the Boltzmann AS mechanism is used to determine which Q-values to exploit from the other agents.

The Pure SVS method does not equally weight the contributing Q-values of the other agents and the agent employing the Pure SVS method. Rather, an action is chosen according to the probability assigned by the Boltzmann AS method, which, for the variables chosen in this research, weight the higher Q-values with higher probabilities.

Similar to the MEC and HECA methods, this method is useful for scenarios where other agents have varying levels of expertise within their Q-tables. Another agent may have no experience with regard to (s,a) but have lots of experience with (s',a'). The MEC method would be able to successfully deduce that the (s,a) Q-value should not be used in the resultant Q-value but that the (s',a') Q-value should, since it provides beneficial information.

Exploiting the knowledge of the other agents using this method would never hinder learning performance, even if the the other agents were somehow regressing in terms of learned knowledge. This method provides a built-in means for determining whether another agent's Q-table will be beneficial or not, and therefore this method may be used in environments where there is no level of expectation regarding the quality of the other agent's policies beforehand.

Average Action Value (AAV) Method

This algorithm and the following one are essentially the pure SVS method with added heuristics. These heuristics alter the type and quantity of information available to the agent for making decisions at each time step. As we scale the number of agents in the simulation, the number of policies in M also increases. When there are 8 agents, for instance, Boltzmann AS must choose between (8 agents) $\times$ (4 actions per agent) = 32 actions at each time step.

The AAV method reduces the number of actions available for the Boltzmann AS to choose from. The heuristic function associated with this method reduces the total number of actions available by generating a single value that serves as a policy representation value for each policy in M. The AAV policy representation heuristic function is shown in Equation (5.3). The AAV heuristic function takes the average value of the Q-values associated with all actions $A(s)$ when the agent is in state s. This average value is the policy representation value using this heuristic.

$$AAV(\pi_n, s) = |A(s)|^{-1} \sum_{a \in A(s)} Q^{\pi_n}(s, a) \text{ for } s \in S, a \in A(s), \pi_n \in M. \tag{5.3}$$

We propose 2 variants each for the AAV and the HAV heuristic functions (the HAV heuristic function will be discussed shortly). The first variant for the 2 methods uses the Boltzmann AS method while the second variant uses an Optimal AS method. When using the Boltmzann AS method with either the AAV or HAV heuristic functions, the policy representation values are used as input to determine probabilities of policy selection. Once a policy is chosen, Boltzmann AS is again used to select an action within that policy. The Optimal AS variant ensures the agent will pick the policy with the highest policy representation value (ties are broken randomly). Similarly, once a policy is chosen for this variant, Boltzmann AS is used to select an action within that policy. The 2 variants serve to allow us to understand how a combination of exploration and exploitation with regard to policy selection compares to simple exploitation. The Boltzmann AS method for policy selection with the AAV and HAV heuristic functions allows for exploration and exploitation while the Optimal AS method purely allows for exploitation.

The AAV method was designed with the intent that it allows for an agent to deduce which policy to defer to for an AS choice because it can determine which policy for that state is, on average, the most developed at the state level. The idea is that a more developed policy at the state level should indicate a well-rounded learner for that state and should provide a better contribution than other less experienced learners that don't have the highest (s,a) Q-value averages. This method works well for scenarios where there are multiple best paths to take in the environment, as would be indicated by similar Q-values for each action in the state.

Highest Action Value (HAV) Method

The heuristic function associated with this method reduces the total number of actions available by generating a single value that serves as a policy representation value for each policy in M. The HAV policy representation heuristic function is shown in Equation (5.4). The HAV heuristic function takes the highest value of the Q-values associated with all actions $A(s)$ when the agent is in state s. This highest value is the policy representation value using this heuristic.

$$HAV(\pi_n, s) = \max_a Q^{\pi_n}(s, a) \text{ for } s \in S, a \in A(s), \pi_n \in M. \tag{5.4}$$

The HAV method was designed with the intent that it allows for an agent to deduce which policy to defer to for an AS choice because it can determine which policy for that state is the most developed at the state-action level. Ideally, this method improves upon the granularity and accuracy of the policy AS because it relies on the highest Q-value in the state rather than an average, which could be less effective in certain situations. For instance, if there is a clear, single best action to choose in a state (as indicated by a single high Q-value), the Q-values of the other actions in the state should not matter as much. However, if another

agent has explored the other actions more thoroughly, there is a chance that these Q-values could be higher. The AAV method would declare that the other agent has the best policy to choose from, but in actuality, the agent with the highest Q-values for the correct action in that state has a more accurate Q-value and should instead be chosen. The HAV method would correctly identify the best policy to select from in this situation, whereas the AAV method would not necessarily do so.

5.2 Non-Reciprocating Methods Results

5.2.1 Experiments

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 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 Q-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 Q-values to change between time steps. Note that with each successive experiment, agents $[A_1, ..., A_{N-1}]$ are employing a reference method with higher learning performance than the one before. These unique environments allow us to explore the following questions:

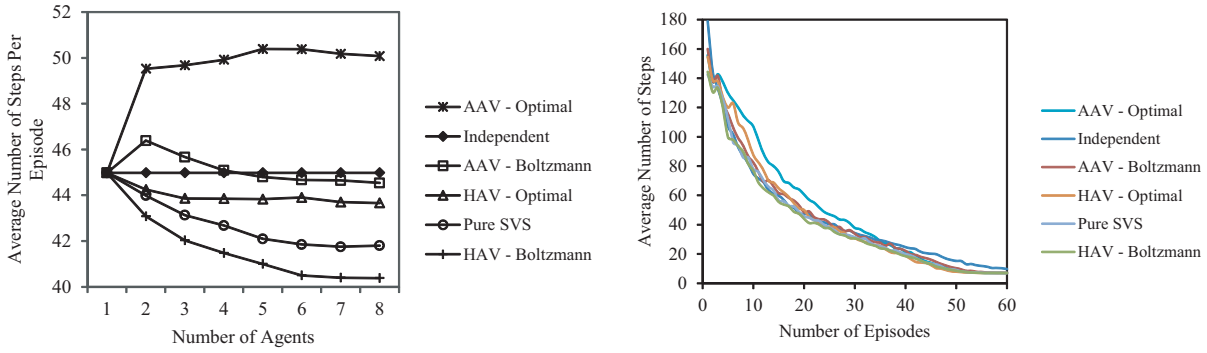
1) How do the proposed methods perform in environments where other agents do not reciprocate using the same method?

2) How does the learning performance of the proposed methods compare to the learning performance of the (reciprocating) reference methods?

3) How does scaling the number of agents in the environment affect the learning performance of the non-reciprocating methods?

4) How do the proposed method's learning performances fare when faced with single or multiple Q-value changes per policy per time step?

5) How do the proposed method's learning performances fare when all other agents in the environment have policies that synchronize after each time step?



(a) Results for A_0 as the number of agents varies.

(b) Results for A_0 when there are 8 agents.

Figure 5.2: Simulation results for A_0 using the extended AS methods while the other agents use the Independent sharing method.

Experiment 1

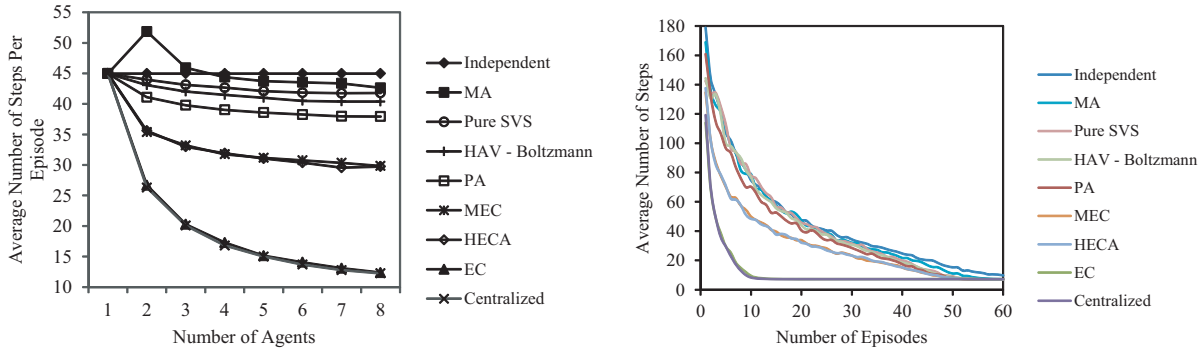
In this experiment, we explore how the proposed algorithms perform for A_0 when agents $[A_1, ..., A_{N-1}]$ are using the Independent sharing method. We first explore the extended AS methods: pure SVS, AAV – Boltzmann, AAV – Optimal, HAV – Boltzmann, and HAV – Optimal. Results are shown in Fig. 5.2a and Table 5.1. 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 agents are able to take advantage of more knowledge accumulated by other agents learning in the same environment. Table 5.1 shows that the pure SVS and HAV – Boltzmann methods significantly outperform the other methods, including the Independent learning reference method. Originally designed for RL environments with positive reward systems, the optimal variants of the AAV and HAV methods exhibit learning performance close to, or worse than, the learning performance of the Independent method.

The results demonstrate three main points for the pure SVS and HAV – Boltzmann extended AS methods: 1) A_0 is able to simultaneously further develop its own policy in addition to exploiting others' policies. This is made evident because overall performance improves and does not stagnate. 2) The results also demonstrate that the learning performance improves when agents are able to exploit others agents' policies. By simply including all cooperating agents' policies in M and expanding the scope of the Boltzmann AS method to include the actions in these policies, A_0 is able to determine when to exploit the other policies to their benefit. 3) Lastly, the performance increase associated with our method demonstrates that A_0 learns to balance exploitation of its own policy, those policies of the other agents, and continue to explore the environment through random exploratory actions.

We now explore the results for the regular AS methods. We carry forth the top 2 performing algorithms, pure SVS and HAV – Boltzmann, from the extended AS methods. Results are shown in Fig. 5.3a.

Table 5.1: Simulation results for A_0 using the extended AS methods while the other agents use the Independent sharing method.

Sharing Method	Average Steps with 95% CI
AAV - Optimal	50.08 ± (0.52)
Independent:	44.98 ± (0.33)
AAV - Boltzmann	44.54 ± (0.38)
HAV - Optimal	44.15 ± (0.49)
Pure SVS	41.80 ± (0.34)
HAV - Boltzmann	40.38 ± (0.31)



(a) Results for A_0 as the number of agents varies.

(b) Results for A_0 when there are 8 agents.

Figure 5.3: Simulation results for A_0 using the regular and best 2 AS methods while the other agents use the Independent sharing method.

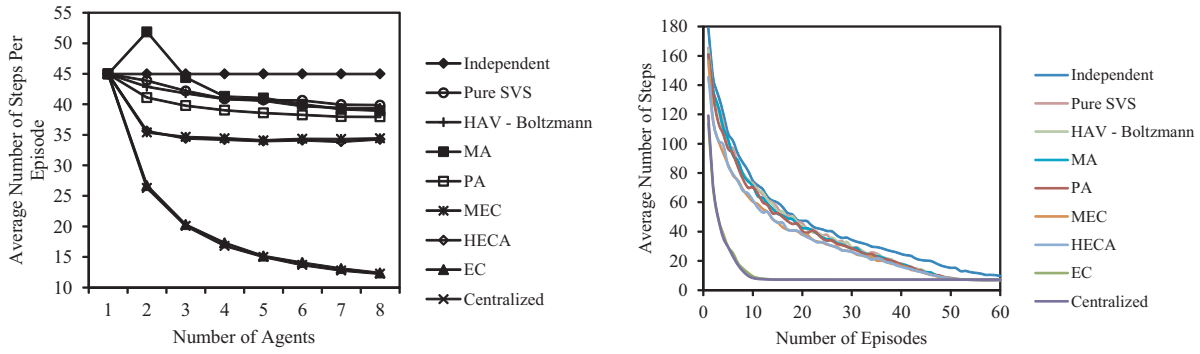
Table 5.2: Simulation results for A_0 using the regular and best 2 AS methods while the other agents use the Independent sharing method.

Sharing Method	Average Steps with 95% CI
Independent	44.98 ± (0.33)
MA	42.64 ± (0.37)
Pure SVS	41.80 ± (0.34)
HAV – Boltzmann	40.38 ± (0.31)
PA	37.94 ± (0.30)
MEC	29.84 ± (0.21)
HECA	29.74 ± (0.20)
EC	12.37 ± (0.11)
Centralized	12.24 ± (0.11)

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 5.2 shows that, according to t-tests, the differences in performance among all listed methods, excepting the MEC and HECA methods, are statistically significant. Fig. 5.3b illustrates how all agents ultimately converge to the same policy, but vary according to learning performance rates.

The MA method performs more poorly than the PA method. This is expected as the PA method operates with all agents participating in synchronizing to the same Q-table after each time step t, thus allowing agents to make an action-selection during their next step using the same Q-table as one other. The MA method operates with A_0 having a Q-table that is not synchronized with the other agent's Q-tables and learning performance goes down as a result. Q-values can conflict with one another and cause harmful learning effects. Note that there is a negative performance spike for the case when there are 2 agents in the environment, where A_0 is learning with the MA method and A_1 uses Independent learning. The reasoning is as follows: consider the case where A_0 has experienced (s,a) at time step t and A_1 has not. A_0 has more beneficial knowledge due to direct experience with (s,a) and A_1 does not. When determining the resultant Q-value for A_0 using the MA strategy, A_1's Q-value contribution will be just as significant as A_0's Q-value contribution. After each successive time step where no updates occur on the (s,a) Q-value, the resultant (s,a) Q-value will continue to be averaged with A_1's outdated Q-value. This guides A_0's Q-value for (s,a) away from the valid value that A_0 initially set for this state-action Q-value after directly experiencing it. This indicates that for this case and, to a lesser extent, the case where there are 3 agents, that it is more beneficial to use the Independent learning method. However, the MA method outperforms Independent learning as we increase the number of agents. When there are 3 or more agents for A_0 to interact with, this trend holds true.

The performance of the MEC and HECA methods are not significantly different. This result indicates that modifying the MEC method to average the Q-values associated with the tied experience values does not yield any learning performance gains. Both methods outperform



(a) Results for A_0 as the number of agents varies.

(b) Results for A_0 when there are 8 agents.

Figure 5.4: Simulation results for A_0 when the other agents use the PA sharing method.

the Independent, MA, and PA methods but underperform the EC and Centralized methods. Similar to the reasoning why the reciprocating PA method outperforms the non-reciprocating MA method, the non-reciprocating MEC method cannot synchronize its Q-table and E-table values with the other agents. Therefore, action-selection decisions are made by each agent operating with potentially different Q-tables. Because the Q-tables are not synchronized, the agents learn at a slower pace than they would using the EC method, but still outperform the Independent method by more than 15 average steps per episode.

The results of this experiment indicate that it is possible for an agent to improve learning performance in environments where other agents do not reciprocate with it. This experiment shows that the MEC and HECA methods perform well when the other agent's policies have different values and do not synchronize to the same policy after each time step t.

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. 5.4a. Table 5.3 shows that, according to t-tests, the differences in learning performance among all methods, again excepting the MEC and HECA methods, are statistically significant. For this experiment, the MEC and HECA methods perform well when the other agent's policies have similar values and synchronize to the same policy after each time step t. The PA method outperforms the MA method, but to a lesser extent when compared to the results from experiment 1. The pure SVS and HAV – Boltzmann methods both perform better as well, when compared to the results from Experiment 1. Fig. 5.4b shows the learning performance rates associated with this experiment for A_0.

Overall, the experiment indicates that the MA, pure SVS, and HAV – Boltzmann method's performances improved compared to the results in Experiment 1. Note that the performances of the MEC and HECA methods became worse, however, they still performed the best

Table 5.3: Simulation results for A_0 when the other agents use the PA sharing method.

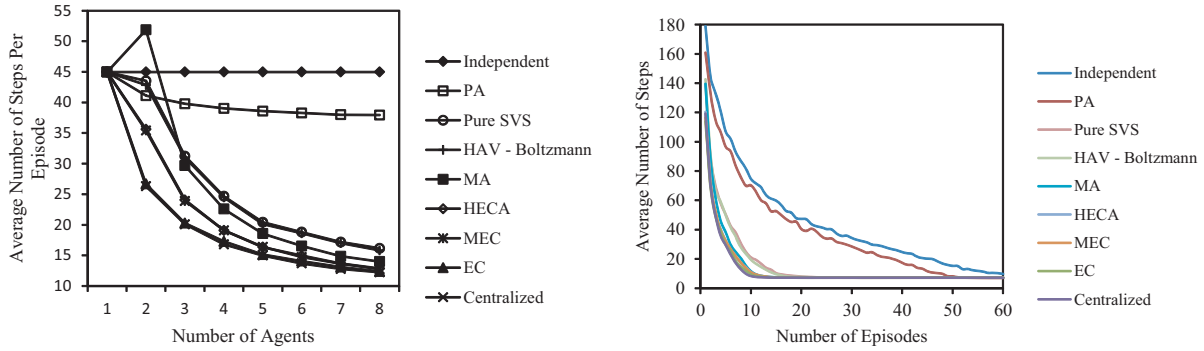
Sharing Method	Average Steps with 95% CI
Independent	44.98 ± (0.33)
Pure SVS	39.87 ± (0.30)
HAV – Boltzmann	39.37 ± (0.27)
MA	38.94 ± (0.34)
PA	37.94 ± (0.30)
MEC	34.42 ± (0.26)
HECA	34.27 ± (0.26)
EC	12.37 ± (0.11)
Centralized	12.24 ± (0.11)

out of the 5 methods we propose. Both experiment 1 and 2 confirm that the proposed methods can suitably adapt to and perform during both types of situations – synchronized and unsynchronized Q-tables – encountered for sharing strategies. Additionally, the methods perform well when faced with single or multiple Q-value changes within the Q-tables per time step.

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. 5.5a. Table 5.4 shows that, according to t-tests, the differences in performance among all listed methods, excepting the MEC and HECA methods, are statistically significant. The learning performance trends for this experiment differs between the PA and MA methods. The MA method significantly outperforms the PA reference method. This is a result caused by the other 7 agents employing the EC method and thus providing access to improved Q-tables for the MA method at each time step. Similar to the results from Experiment 2, the results for this experiment show that the learning performances for the pure SVS and HAV – Boltzmann methods perform better than the Independent method, but of all non-reference methods, perform the worst. Fig. 5.5b shows the learning performance rates associated with this experiment for A_0.

All non-reference 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 MEC and HECA methods perform well when the other agent's policies have similar values and synchronize to the same policy after each time step t.



(a) Results for A_0 as the number of agents varies.

(b) Results for A_0 when there are 8 agents.

Figure 5.5: Simulation results for A_0 when the other agents use the EC sharing method.

Table 5.4: 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)
Pure SVS	16.16 ± (0.18)
HAV – Boltzmann	15.81 ± (0.17)
MA	13.99 ± (0.17)
HECA	12.86 ± (0.12)
MEC	12.75 ± (0.12)
EC	12.37 ± (0.11)
Centralized	12.24 ± (0.11)

5.2.2 Summary

Results from the 3 experiments show that it is possible for an agent to improve learning performance in environments where other agents do not reciprocate with it. The top 5 performing methods we proposed – pure SVS, HAV – Boltzmann, 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. The proposed methods can suitably adapt to and perform during both types of situations – synchronized and unsynchronized Q-tables – encountered for sharing strategies. Additionally, the methods perform well when faced with single or multiple Q-value changes within the Q-tables per time step. Most notably, the MEC and HECA methods perform best overall across the 3 experiments, with no significant difference in learning performance amongst the 2 methods.

5.3 Alternative Non-Reciprocating Methods Approach

In the last section, our experiments reveal that the MA method achieves worse performance than Independent learning for the case of 2 agents. Our previous discussion on the matter concludes with our belief that the agent employing the MA method was not assimilating the shared knowledge correctly. In this section, we propose an alternative agent structure and minor modification to the MA method to allow this updated agent structure to work properly. Additionally, we apply the updated agent structure to the MEC and HECA methods in an attempt to determine if it will also improve the learning performance for these two methods as well. Note that we do not apply and test the updated agent structure to the pure SVS and HAV – Boltzmann methods because these 2 methods do not assimilate the policy data gathered from $\{\pi_0, ..., \pi_{N-1}\}$ into a policy that will be assigned to π_0; therefore, they are not applicable.

5.3.1 Agent Structure

Whereas the previous agent structure assimilates learned knowledge from other agents back into the agent's self policy, π_0, this new structure is designed to separate the two. As seen in the following definition, this new structure adds one additional policy to the agent's memory in order to provide the memory for this design to be possible.

Definition 2 *An agent's memory, M, is a set of $(N+1)$ policies $\{\pi_0, ..., \pi_N\}$ 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. π_N is an additional policy used to store assimilated agent data.*

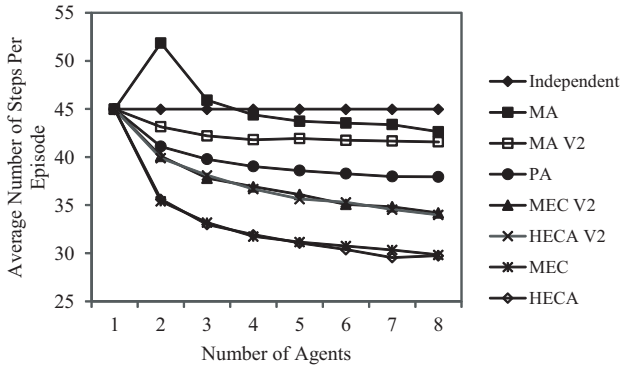


Figure 5.6: Simulation results for A_0 – With Algorithm Modifications.

5.3.2 Method Modifications, Description and Design

The MA, MEC, and HECA methods are modified to make use of the alternative agent structure. To distinguish between the original methods, we denote the newer methods by appending the 'V2' modifier. The MA V2 method is similar to the MA method except that after averaging all agents' policies, π_N is assigned the averaged Q-table, instead of π_0. Additionally, when the agent is making an action-selection, it uses the π_N policy to make a decision but subsequently updates the π_0 policy to record direct agent experience with the environment. The benefit of this design over the previous design is that the MA V2 method averages the policies $\{\pi_0, ..., \pi_{N-1}\}$ and assigns the resultant Q-table to π_N, therefore alleviating the problem of having the next averaged policy incorporate any previously averaged policy data. Modifying the MEC and HECA methods is a similar process. The only difference when compared to the MA V2 method is that the MEC V2 and HECA V2 methods use the experience-counting and experience-counting with averaging methods, respectively, on policies $\{\pi_0, ..., \pi_{N-1}\}$ to determine the resultant policy that is assigned to π_N.

5.4 Alternative Non-Reciprocating Methods Results

For this experiment, we seek to determine if the alternative agent structure proposed and the subsequent modifications to the 3 learning methods have any effect on learning performance for the agents when compared to the previous agent structure and learning methods. The domain we conduct the experiment in and the evaluation techniques we used previously are the same for this experiment to ensure consistency. For each test, agent A_0 utilizes one of our learning methods and agents $\{A_1, ..., A_{N-1}\}$ make use of the Independent learning method. Fig. 5.6 displays the results for this experiment along with the results from the first experiment in the previous section where agents $\{A_1, ..., A_{N-1}\}$ use the Independent learning method. Additionally, we also include 2 reference methods, Independent and PA, to provide reference to indicate how the various methods compare with standard methods.

Table 5.5: Simulation results for A_0 – With Algorithm Modifications.

Sharing Method	Average Steps with 95% CI
Independent	44.98 ± (0.33)
MA	42.64 ± (0.37)
MA V2	41.58 ± (0.28)
PA	37.94 ± (0.30)
MEC V2	34.20 ± (0.25)
HECA V2	33.99 ± (0.24)
MEC	29.84 ± (0.21)
HECA	29.74 ± (0.20)

Most notably, the results indicate that the MA V2 method performs better than the MA method. For the case when there are 2 agents in the environment, the MA V2 method has better performance than the Independent learning method. This indicates that the agent architecture and minor algorithm redesign that make up MA V2 correct the problems associated with the MA method. The resultant Q-table that is assigned to π_0 for the MA method after each step is negatively impacting learning performance. By creating π_N and assigning the resultant policy after each step to this policy, the method's learning performance improved.

Unfortunately, the good results do not extend to the MEC V2 and HECA V2 methods. Table 5.5 shows that, according to t-tests, the differences in performance among MEC and HECA and their updated versions are statistically significant. The updated versions of the methods perform more than 14% more poorly learning performance-wise. The alternative agent structure is bad for performance as it requires the agent to store the resultant policy from the averaging operation in π_N and perform action-selection on it, but update π_0 with direct step experience. This mismatch does not follow the Q-Learning algorithm of performing both action-selection and updates of direct step experience for the same policy and degrades learning performance as seen in MEC V2 and HECA V2. The reason this alternative agent structure appears to improve performance when compared to the MA method is that the problem with MA and its reuse of the averaged resultant policy into the next resultant policy is so degrading on performance that once this problem is removed, as is done in MA V2, learning performance improves.

The design for the MEC V2 and HECA V2 methods was thoroughly explored before testing. The first design involved only using agent A_0's π_0 policy for both the action-selection and direct step experience update. This approach does not utilize the knowledge of the other agents. The next design involved using the resultant policy π_N for action-selection but updating π_0 with the direct step experience gained. This design was tested in the experiment and suffers from poor performance. Another permutation for design with this agent structure was to use the π_N policy for storing the resultant policy and action-selection, but to update both itself and the π_0 policy with the step experience it gains. Lastly, a design could

allow storing the resultant policy, action-selection, and direct step experience updates to π_N, entirely bypassing π_0. These last 2 designs reduce to the design we use for the original MEC and HECA algorithms because it is more effective, both learning performance-wise and memory-wise, to enable π_0 to store the resultant policy, be used for A_0's action-selection, and for A_0 to append step experience updates.

This experiment indicates that the alternative agent structure and its associated MA V2 method perform better than the original MA method. Therefore, the remaining work in this paper will use this strategy. Henceforth, the MA V2 method and its agent structure will replace the original MA method and agent structure and will simply be referred to as the MA method. For the MEC and HECA methods, we will be using the originally proposed methods as the alternative versions did not perform well.

Chapter 6

Limited Communication Range

6.1 Limited Communication Range Approach

In the previous chapter, we considered non-reciprocating sharing methods in cooperative learning environments in which agents were given the ability to communicate with all other agents in the environment with no restrictions on communication range. In this section, we consider agents with a limited communication range in an attempt to recreate a simulation of physically embodied agents learning a navigation task. We seek to determine how the previously presented non-reciprocating sharing methods perform in such an environment and we explore the effects of scaling communication range on learning performance and communication rates. Additionally, this is an attempt to move away from the unrestricted communication problem which Stone and Veloso compare to as being similar to the single-agent system case [46].

6.2 Limited Communication Range Results

6.2.1 Experiments

Communication frequencies measure how often a Q-table is transferred from one agent to another. For example, if A_0 reads the Q-table stored by A_1, the communication frequency would increase by 1 communication unit as 1 Q-table was accessed and had to be transferred from one agent to another. Because our work concerns non-reciprocating methods, communication frequency will only be recorded for A_0 since it employs our proposed methods. Any type of inter-agent communication will always be between A_0 and $\{A_1, ..., A_{N-1}\}$, indicating that A_0 has accessed one of the other agent's policies. The communication range indicates how many cells in each direction the agent is allowed to search for other agents to commu-

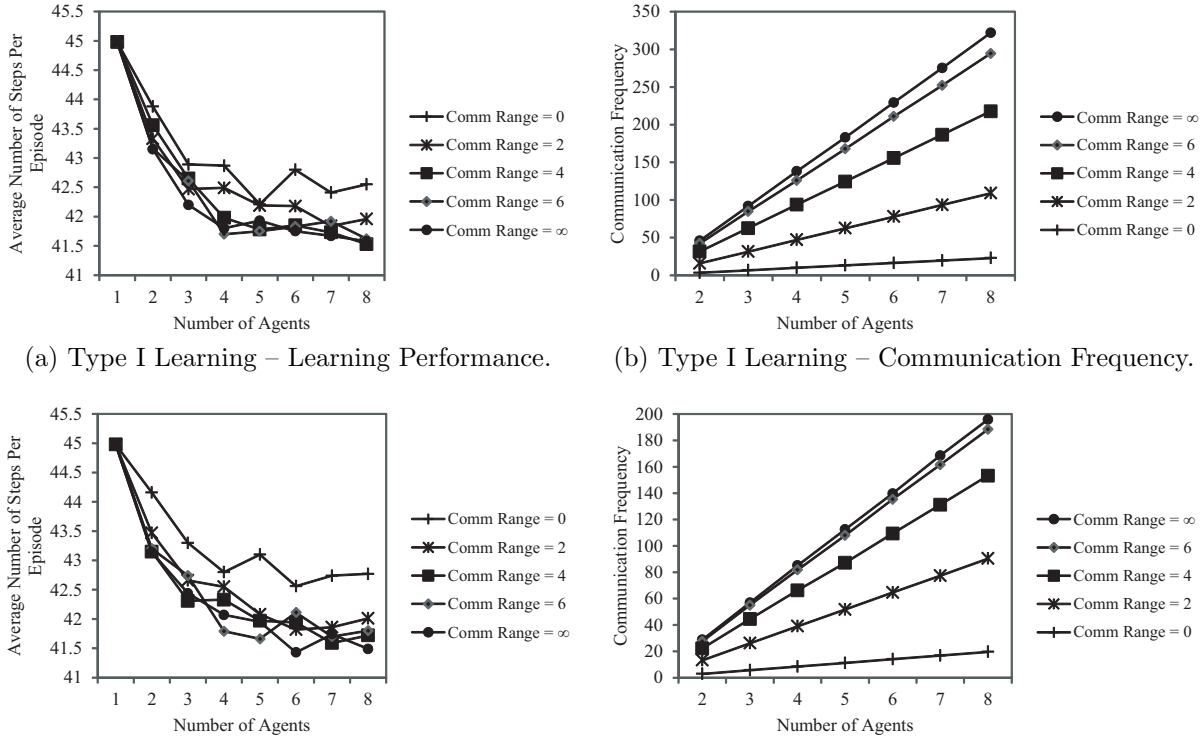
nicate with. A communication range of 0 indicates the agent can only communicate with other agents within its current cell and a communication range of 8, for an 8x8 map, would indicate communication across the entire map. Agents cannot communicate through walls. Whenever the agent can communicate with another agent, it chooses to do so.

In these experiments we introduce the concepts of Type I and Type II learning. Type I learning allows an agent to continue to learn, even after episode completion. Once an agent employing Type I learning has finished an episode and is in the goal position, it may continue to communicate with active agents in the environment within its communication range. Note that an active agent is one that is defined as an agent that has not completed the current episode. If an agent has completed the current episode, an agent employing Type I learning will not communicate with it. Realistically, this makes sense as physically embodied agents will still be in the goal state while the other agents finish. Therefore it makes sense to allow agents to continue to communicate and to attempt to improve upon their learning policies. An alternative to this would be to allow Type I learning agents to communicate with all agents within communication range, regardless of agent episode completion status, until all agents have completed the current episode. Both ways of learning would fairly and adequately test our methods; we chose to use the former. Type II learning does not allow an agent to continue learning through inter-agent communication once the agent's current episode is complete. For these experiments involving communication frequency analysis we present results for both types of learning. This will enable us to better understand the impact of both types of learning on learning performance and communication frequency. All previous experiments in this work involve Type I learning agents.

We present 3 experiments, where the first focuses on the MA method, the second on the MEC method, and the third on the HECA method. Although the MEC and HECA methods perform equally well in our previous tests and outperform the MA method, for sake of completeness we also test the MA method as this more realistic simulation setting may yield different results. Note that we do not include separate experiments for the pure SVS and HAV – Boltzmann methods because the resulting trends are similar to the 3 experiments we will present. We do, however, include results from testing of these 2 methods in a limited communication environment in the Results section at the end of this section in Table 6.4. For each experiment we record the learning performance and communication frequencies as we scale the simulation from 1 to 8 agents. For these experiments, $\{A_1, ..., A_{N-1}\}$ employ the Independent learning method.

Experiment 1

For this experiment, A_0 employs the MA method, while the other agents employ the Independent learning method. The results shown in Fig. 6.1a and Fig. 6.1b are for Type I learning and indicate, intuitively, that as the communication range decreases and becomes more limited, learning performance degrades and communication frequency decreases. Although the



(a) Type I Learning – Learning Performance.

(b) Type I Learning – Communication Frequency.

(c) Type II Learning – Learning Performance.

(d) Type II Learning – Communication Frequency.

Figure 6.1: Limited Communication Tests for the MA Method.

learning performance rates exhibit unstable curves, the overall trend is clear. This is especially apparent in Table 6.1 as this table shows that, according to t-tests, the differences in learning performance for 8 agents when the communication rate is infinite and when the communication rate is 0 are statistically significant. For the communication frequency graph there exists a linear trend in which as the number of agents scales up, the communication frequency linearly increases. Table 6.1 also shows that, according to t-tests, the differences in communication frequencies as we vary the communication range are statistically significant.

Results from Type II learning are shown in Fig. 6.1c, Fig. 6.1d, and Table 6.1. According to Table 6.1 the differences in learning performance between Type I and Type II learning are not statistically significant. However, according to the same table, the communication frequencies between Type I and Type II learning are statistically significant. As expected, these results show that Type II learning results in reduced communication frequencies. Once A_0 has reached the goal state and finishes its current episode, the agent will no longer communicate with other agents until the next episode begins. Results analyzing the differences in communication frequencies across the 3 proposed learning methods – the main reason for presenting the Type II learner – are presented in the Overall Results portion of this section.

Table 6.1: Limited Communication Tests for the MA Method.

Comm Range	Average Steps Type I	Average Steps Type II	Comm Frequency Type I	Comm Frequency Type II
∞	41.58 ± (0.28)	41.49 ± (0.28)	321.96 ± (0.86)	195.82 ± (0.84)
6	41.62 ± (0.29)	41.80 ± (0.28)	294.55 ± (0.76)	188.39 ± (0.78)
4	41.53 ± (0.28)	41.72 ± (0.29)	217.69 ± (0.56)	153.26 ± (0.63)
2	41.96 ± (0.30)	42.01 ± (0.30)	108.90 ± (0.30)	90.32 ± (0.33)
0	42.55 ± (0.30)	42.77 ± (0.30)	22.80 ± (0.10)	19.53 ± (0.11)

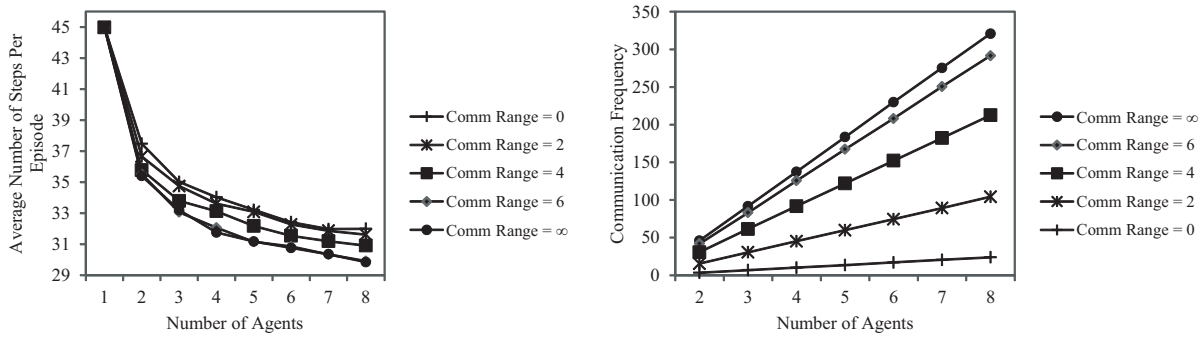
Table 6.2: Limited Communication Tests for the MEC Method.

Comm Range	Average Steps Type I	Average Steps Type II	Comm Frequency Type I	Comm Frequency Type II
∞	29.84 ± (0.21)	30.36 ± (0.21)	320.82 ± (0.85)	168.17 ± (0.78)
6	29.91 ± (0.20)	30.60 ± (0.21)	291.52 ± (0.77)	163.15 ± (0.75)
4	30.91 ± (0.21)	30.79 ± (0.20)	212.51 ± (0.56)	134.18 ± (0.56)
2	31.62 ± (0.21)	31.29 ± (0.21)	104.24 ± (0.28)	81.49 ± (0.31)
0	31.99 ± (0.22)	31.68 ± (0.21)	23.86 ± (0.10)	19.85 ± (0.11)

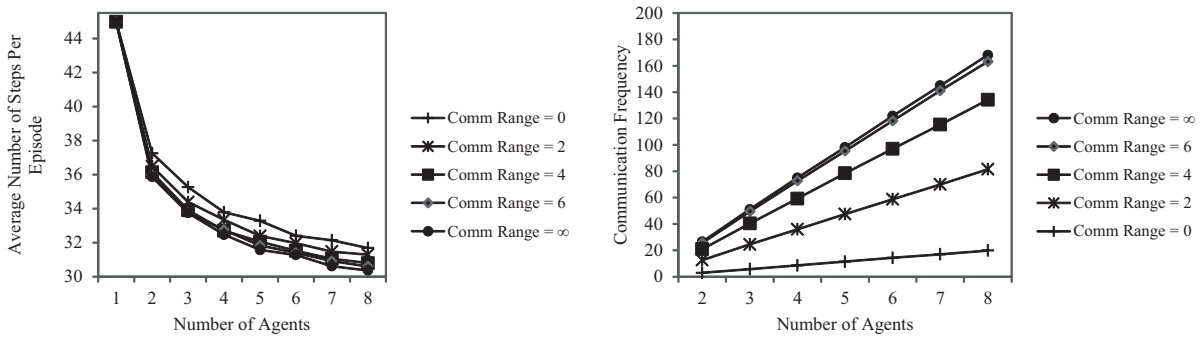
Experiment 2

For this experiment, A_0 employs the MEC method, while the other agents employ the Independent learning method. The results shown in Fig. 6.2a and Fig. 6.2b are for Type I learning and indicate that as the communication range decreases and becomes more limited, learning performance degrades and communication frequency decreases. These results agree with the results from the previous experiment. One difference we note arises with the learning performance graph in which the learning performance rates exhibit stable curves. Table 6.2 indicates that varying the communication range produces results that are statistically significant. Again, as we scale the number of agents in the simulation environment, the communication frequencies scale linearly.

Results from Type II learning are shown in Fig. 6.2c, Fig. 6.2d, and Table 6.2. According to Table 6.2 the differences in learning performance between Type I and Type II learning are statistically significant. Fig. 6.3 provides a visual depicting learning performance for both Type I and Type II learning for the MA and MEC methods. As previously stated, the MA method's learning performance across Type I and Type II learning does not change. For the MEC method, as the communication range decreases for both Type I and Type II learning, learning performance degrades because A_0 less frequently contacts other agents and updates their respective Q-table with A_0's memory. The rate at which Type I learning degrades is more pronounced than that of Type II learning. When the comm range is 4, the learning performance for the Type I learning matches that of Type II learning and after this point, Type I performs more poorly than Type II. The differences when the communication ranges are all but 4 are statistically significant.



(a) Type I Learning – Learning Performance. (b) Type I Learning – Communication Frequency.



(c) Type II Learning – Learning Performance. (d) Type II Learning – Communication Frequency.

Figure 6.2: Limited Communication Tests for the MEC Method.

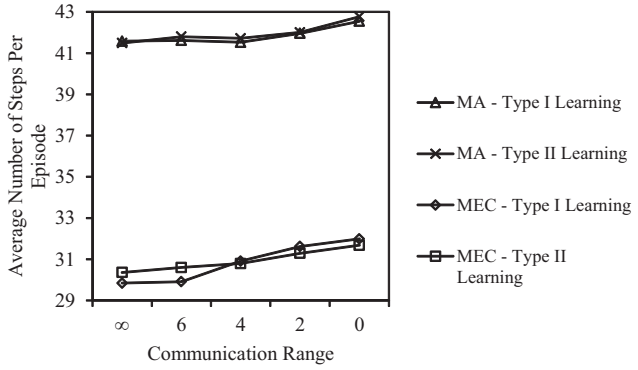


Figure 6.3: Type I vs. Type II Learning for the MA and MEC Methods.

For the case where the communication range is ∞, it is logical for Type I learning to have better learning performance because A_0 will continue to improve its policy via communicating with other agents after finishing the current episode. Whereas when A_0 is employing Type II learning, once it is finished with its episode it is done learning until the next episode. As the comm range approaches 0, Type II learning performs better than Type I learning. Analysis of the results leads us to believe that this is a side effect of the MEC method due to the

Table 6.3: Limited Communication Tests for the HECA Method.

Comm Range	Average Steps Type I	Average Steps Type II	Comm Frequency Type I	Comm Frequency Type II
∞	29.74 ± (0.20)	30.19 ± (0.20)	321.50 ± (0.85)	167.69 ± (0.75)
6	30.03 ± (0.21)	30.45 ± (0.21)	291.48 ± (0.74)	163.00 ± (0.73)
4	30.78 ± (0.21)	30.62 ± (0.20)	212.00 ± (0.55)	134.31 ± (0.57)
2	31.55 ± (0.21)	31.22 ± (0.22)	103.95 ± (0.29)	81.60 ± (0.31)
0	31.78 ± (0.23)	31.64 ± (0.23)	23.91 ± (0.10)	19.76 ± (0.11)

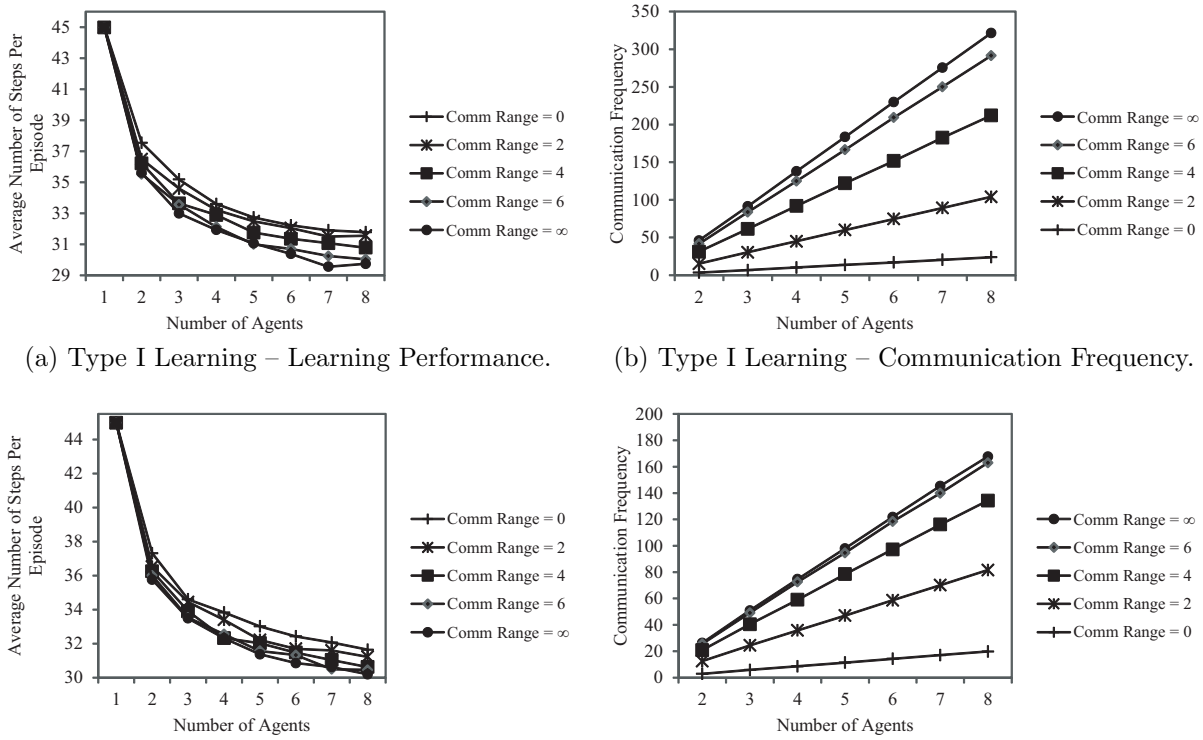
way experience is counted. As previously stated, the MEC method rewards experience by detecting changes in policy Q-values from one time step to the next. When communication range is more limited, A_0 may go for longer periods of time without contacting the other agents. Once communication occurs, only a single experience point will be rewarded for Q-value differences that could be small or large – no differentiation of the size of this value occurs. Therefore, over time A_0's E-tables distort the truth about which agents actually have more experience than one another. Results indicate that the extended communication following a completed episode for lower communication ranges degrades performance more than it helps. We arrive at this conclusion because the only difference between Type I and Type II learning is that Type I learning will continue to learn from the other actively learning agents in the environment. For the MA method, the differences are not statistically significant. We believe the reason for this to be because the MA method does not rely on counting experience and therefore gaps with a lack of communication does not impact the method the way that the MEC method is affected.

Experiment 3

For this experiment, A_0 employs the HECA method, while the other agents employ the Independent learning method. The results shown in Fig. 6.4a, Fig. 6.4b, 6.4c, 6.4d, and Table 6.3 are nearly identical to the results from the experiment with the MEC method. These results again reinforce that the MEC and HECA methods perform nearly identically in terms of learning performance rates. It is also confirmed that the MEC and HECA methods perform nearly identically in terms of communication frequency.

6.2.2 Results

All 3 experiments confirm the overall trends that as we decrease the communication range among agents, learning performance degrades and communication frequency decreases. A side-by-side comparison of the learning performance rates and communication frequency rates for each of the 5 methods – we now include results from the pure SVS and HAV – Boltzmann methods – with both Type I and Type II learning as we decrease the com-



(a) Type I Learning – Learning Performance.

(b) Type I Learning – Communication Frequency.

(c) Type II Learning – Learning Performance.

(d) Type II Learning – Communication Frequency.

Figure 6.4: Limited Communication Tests for the HECA Method.

munication range is presented in Table 6.4. To accommodate the table on the page, we abbreviated the pure SVS and HAV – Boltzmann methods as SVS and HAV, respectively. These results indicate that the MEC and HECA methods significantly outperform the pure SVS, MA, and HAV – Boltzmann methods in terms of learning performance, even in an environment where communication is limited. The communication frequency results in Table 6.4 for Type I and Type II learners indicate that the communication frequencies for the MEC and HECA methods are lower than the communication frequencies associated with the MA method. This result is in agreement with the notion that a better learning rate is associated with a lower communication frequency – excepting when the communication range is 0, as noted below.

For the case where the communication range is 0 for both Type I and Type II learners, the difference between the MA method's learning performance and that of MEC or HECA is statistically significant. To understand this finding, we graphed the results from the MEC method with Type II learning as the communication range varied for 8 agents, as shown in Fig 6.5a. All communication range curves start at a higher communication frequency than they converge to, except for when the communication range is 0. An alternative, more detailed view of this finding is shown in Fig. 6.5b, where we graph the communication frequency vs. number of episodes for 2 through 8 agents. It is clear from these two graphs

Table 6.4: Limited Communication Tests for the Pure SVS, MA, HAV – Boltzmann, MEC, and HECA Methods, A Comparison.

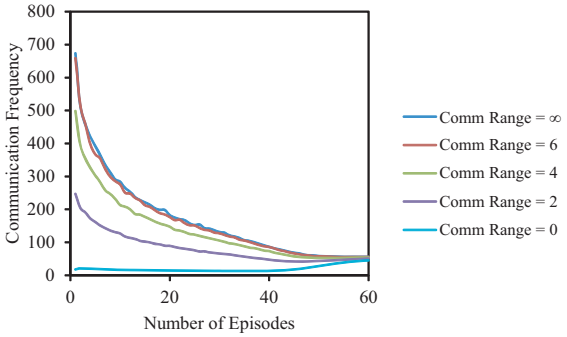
Comm Range	Method	Average Steps Type I	Average Steps Type II	Comm Freq. Type I	Comm Freq. Type II
∞	SVS	41.80 ± (0.34)	42.02 ± (0.34)	321.21 ± (0.85)	195.43 ± (0.86)
	MA	41.58 ± (0.28)	41.49 ± (0.28)	321.96 ± (0.86)	195.82 ± (0.84)
	HAV	40.38 ± (0.34)	40.39 ± (0.29)	321.78 ± (0.85)	191.79 ± (0.82)
	MEC	29.84 ± (0.21)	30.36 ± (0.21)	320.82 ± (0.85)	168.17 ± (0.78)
	HECA	29.74 ± (0.20)	30.19 ± (0.20)	321.50 ± (0.85)	167.69 ± (0.75)
6	SVS	42.25 ± (0.35)	42.14 ± (0.35)	294.19 ± (0.73)	187.61 ± (0.86)
	MA	41.62 ± (0.29)	41.80 ± (0.28)	294.55 ± (0.76)	188.39 ± (0.78)
	HAV	40.51 ± (0.33)	40.56 ± (0.31)	293.50 ± (0.72)	184.71 ± (0.80)
	MEC	29.91 ± (0.20)	30.60 ± (0.21)	291.52 ± (0.77)	163.15 ± (0.75)
	HECA	30.03 ± (0.21)	30.45 ± (0.21)	291.48 ± (0.74)	163.00 ± (0.73)
4	SVS	42.50 ± (0.34)	42.24 ± (0.34)	217.26 ± (0.57)	152.46 ± (0.63)
	MA	41.53 ± (0.28)	41.72 ± (0.29)	217.69 ± (0.56)	153.26 ± (0.63)
	HAV	40.81 ± (0.32)	41.01 ± (0.34)	216.64 ± (0.58)	150.31 ± (0.64)
	MEC	30.91 ± (0.21)	30.79 ± (0.20)	212.51 ± (0.56)	134.18 ± (0.56)
	HECA	30.78 ± (0.21)	30.62 ± (0.20)	212.00 ± (0.55)	134.31 ± (0.57)
2	SVS	42.75 ± (0.35)	42.69 ± (0.34)	109.10 ± (0.31)	90.31 ± (0.33)
	MA	41.96 ± (0.30)	42.01 ± (0.30)	108.90 ± (0.30)	90.32 ± (0.33)
	HAV	41.12 ± (0.33)	41.14 ± (0.34)	108.52 ± (0.29)	89.15 ± (0.34)
	MEC	31.62 ± (0.21)	31.29 ± (0.21)	104.24 ± (0.28)	81.49 ± (0.31)
	HECA	31.55 ± (0.21)	31.22 ± (0.22)	103.95 ± (0.29)	81.60 ± (0.31)
0	SVS	43.41 ± (0.37)	43.77 ± (0.38)	23.75 ± (0.09)	20.38 ± (0.11)
	MA	42.55 ± (0.30)	42.77 ± (0.30)	22.80 ± (0.10)	19.53 ± (0.11)
	HAV	41.84 ± (0.35)	42.05 ± (0.33)	23.82 ± (0.09)	20.21 ± (0.10)
	MEC	31.99 ± (0.22)	31.68 ± (0.21)	23.86 ± (0.10)	19.85 ± (0.11)
	HECA	31.78 ± (0.23)	31.64 ± (0.23)	23.91 ± (0.10)	19.76 ± (0.11)

that the curves undergoes an initial spike in communication frequency. The communication frequency then gradually decreases until approximately 2/3 of the episodes are complete, at which point it sharply rises again for the duration of the episodes. This is consistent for agents with a limited communication range that start out with little knowledge of the environment and wander around gathering knowledge, eventually finding the quickest path during the later episodes. As more agents find the quickest path, communication frequency will increase because the agent will have more surrounding agents to communicate with. The graph in Fig. 6.5c overlays both learning performance and communication frequency curves onto the same graph. This graph indicates that because the MEC method has a better learning performance when compared to the learning performance associated with the MA method, it finds the quickest path through the environment within fewer episodes.

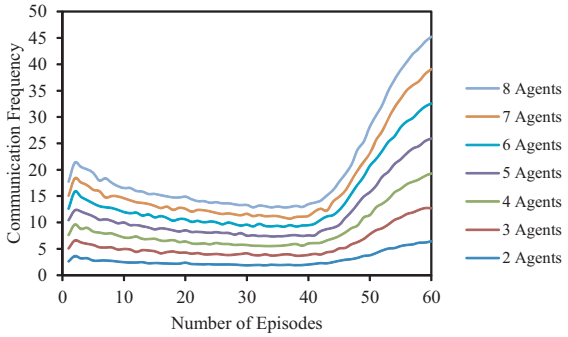
Therefore, initially the communication frequency for this agent will be lower as it stays in the environment for less time because it arrives at the goal state more quickly. As the other agents improve their policies and ultimately find the best path, A_0 will already be on the path and, if the other agents are within range, A_0 will detect them and the communication frequency will increase. As a result of this behavior for communication frequency, the MEC communication frequency average over all episodes will be higher than the communication frequency average for MA. Referring back to Fig. 6.5a, the other communication ranges do not experience this result as the communication frequency rates go down as the number of episodes increase, so a lower communication frequency average indicates a better learning performance due to a quicker episode finish.

The pure SVS and HAV – Boltzmann methods require different reasoning to explain their communication frequency measurements. The results show that while the MEC method has better learning performance when compared to the learning performance for the pure SVS and HAV – Boltzmann methods, the communication frequency average associated with it is lower than the communication frequency average associated with these other 2 methods. Seemingly, this contradicts our previous explanation. However, this is not the case as illustrated in Fig. 6.5c. Notice that the learning curves for both the pure SVS and HAV – Boltzmann methods initially perform worse than the learning curve for the MA method; for approximately the first 25 episodes. At this point, the communication frequencies for these two methods also exceed that of the MA and MEC method's communication frequencies. As aforementioned, this is because the agents employing the pure SVS and HAV – Boltzmann methods are in the environment for a longer period of time and therefore come into more frequent contact with the other agents. Starting at approximately 25 episodes, the learning curves for the 2 methods stay on par with, and then significantly improve when compared to the MA method's learning curve. The learning curves for the 2 methods converge to the learning performance of the MEC method around the 48 episode mark. According to our previous reasoning, this indicates that the communication frequency will go up. In fact, it does go up and manages to align with the same communication frequencies that the MEC method experiences. To summarize, the 2 methods experience higher communication frequencies earlier on when compared to MA and MEC method's communication frequencies. This is followed by even higher communication frequencies by the end of the 60 episodes that align with the communication frequencies attained by the MEC method; this is the reason for higher communication frequency averages by these 2 methods. Therefore, while the learning performance rates for the pure SVS and HAV – Boltzmann methods are worse than the learning performance rates for the MEC method, the communication frequencies are higher for the 2 methods when compared to the MEC method.

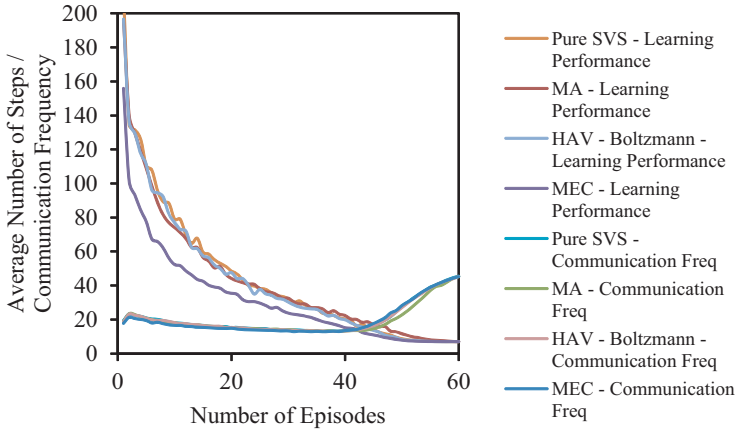
The communication frequency trends are similar for Type I and Type II learners, with the exception being when the communication range is ∞. Recall that Type I learners will continue to learn from other agents that have not reached the goal. Therefore, for all 5 methods, A_0 will be communicating with all active agents at every time step until all agents have finished the episode. Even if an agent finishes before A_0 does, A_0 is able to communicate



(a) MEC Method Variable Comm Range.



(b) MEC Method, Comm Range = 0, Variable Number of Agents.



(c) MEC, MA, pure SVS, and HAV – Boltzmann Methods, Comm Range = 0.

Figure 6.5: Type II Learning Analysis.

with it before it reaches the goal and access its Q-table. Therefore, it doesn't matter how fast the learning performance is for the method, the communication frequencies will be the same at this communication range.

For the sake of uniformity within the other sections, the rest of this work will continue experimenting with agents employing Type I learning.

6.3 Improving the MEC Method Approach

In this section we analyze the flaws associated with our MEC method in environments with limited communication ranges and propose new approaches that improve upon the algorithm.

6.3.1 Problem Description

As previously mentioned, when the communication range becomes more limited the MEC method's performance begins to degrade. Experience counts for the other agents become distorted because inter-agent communication is less frequent and the method does not distinguish how much experience should be awarded for a small Q-value update versus a larger one. The following describes a situation that will frequently occur for agents employing the MEC method in limited communication range environments and highlights the problem we are addressing:

There are 3 agents in the environment: A_0, A_1, and A_2. A_1 has 1 experience point with (s,a) and A_0 has communicated with it and accessed its Q-table at time step 1. A_1 wanders off. During this time, A_1 experiences (s,a) 5 more times, for a total of 6 visits to state s while taking action a. A_1 moves within the communication range of A_0 and A_0 accesses A_1's Q-table. According to the MEC method, A_0 sees that the Q-value for (s,a) has changed, but only awards it 1 experience point in its E-table for (s,a). Instead of (s,a) having 6 experience points, according to A_0 it only has 2. This becomes a problem as more agents get involved. Imagine that A_2 has visited (s,a) 3 times and A_0 has been in constant communication with A_2 the whole time. According to A_0, A_2 would have 3 experience points for (s,a) and A_1 would only have 2 experience points for (s,a). When deciding the Q-value to use, the MEC algorithm would choose to use A_2's Q-value for (s,a) over A_1's Q-value for (s,a) even though A_1's Q-value is clearly better due to more experience. In summary, the MEC algorithm cannot distinguish how many times the Q-value for a particular (s,a) pair has changed since the last (s,a) pair update and therefore will not necessarily be able to update the E-table according to how much experience that an agent has actually gained with that (s,a) pair.

To determine how much this problem affects the overall learning performance for agents employing the MEC method, we ran a simulation where we enabled agents $\{A_1, ..., A_{N-1}\}$ to keep track of their own E-tables while employing the Independent learning algorithm. Whenever an agent came within communication range of A_0, A_0 used the other agent's stored E-table values (and its Q-values) to update its own E-table for that agent in memory. This allows A_0 to have an accurate experience count for the other agents when they come within communication range. Using this test, we are able to isolate how much of a problem the limited communication is for the MEC method. This simulation was tested on an (8×8) map with a communication range of 0. We chose 0 because we wanted very limited communication so as to exacerbate the problem for clear results. The results for this simulation are shown in Fig. 6.6 with confidence intervals calculated in Table 6.5. According to Table 6.5, the difference in learning performance for all 3 curves is statistically significant. These results confirm that when A_0 has the correct E-table values for the other agents, learning performance significantly improves. While the learning performance for the simulated problem fix does not match the learning performance for the MEC method when the communication range is ∞, this is logical because A_0 does not communicate with the other agents as frequently. Therefore, the E-table distortion problem previously described

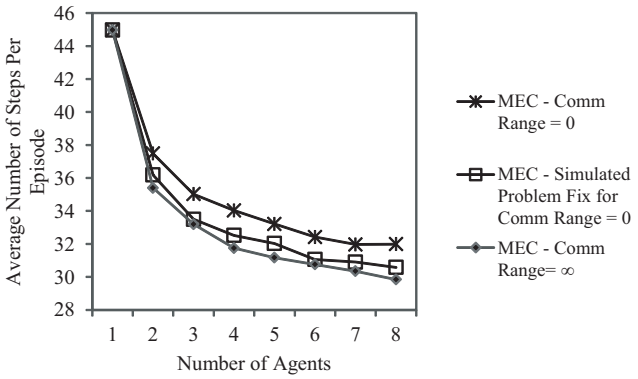


Figure 6.6: Isolating Problem with MEC Method in Limited Communication Environments.

Table 6.5: Isolating Problem with MEC Method in Limited Communication Environments.

Method	Average Steps
MEC - Comm Range = 0	31.99 ± (0.22)
MEC - Simulated Problem Fix for Comm Range = 0	30.58 ± (0.22)
MEC - Comm Range= ∞	29.84 ± (0.21)

has a significant effect on the learning performance for agents employing the MEC method as communication range decreases.

6.3.2 MEC Method Improvement Designs

In an attempt to remedy the problem presented above for the MEC algorithm, we introduce a number of algorithm improvements that take advantage of the difference in Q-values after an update. For the MEC algorithm, a Q-value change from one time step to the next for an agent, as perceived by the agent employing the algorithm, was evaluated with 1 experience point for that particular (s,a) pair. In the following improvements, A_0, or the agent employing the MEC method, will keep track of the average difference in Q-value updates as they occur to π_0 (and in some cases for initialization purposes, $\{\pi_1, ..., \pi_{N-1}\}$) in A_0's memory. The basic idea is that when the agent needs to evaluate how many experience points to assign to an (s,a) pair after an update, A_0 will simply divide the difference in Q-values from the most recent update by the value that contains the average difference in past Q-value updates. The resultant value will the number of experience points to assign to the (s,a) pair in the E-table for that agent. This simple idea is expanded below to take advantage of a number of different implementation techniques to determine the most beneficial method for improving the MEC method – with the goal being to eliminate the distorted E-table problem for environments with lower communication ranges.

The 3 methods we present take different approaches to updating the value that contains the

Algorithm 2 Updating UD after Direct Experience

Input: $UD, qValue_{new}, qValue_{cur}, s_x, s_y, a, agent$
Output: UD
1: **if** $qValue_{new} \neq qValue_{cur}$ **then**
2: Initialize $diff \leftarrow qValue_{new} - qValue_{cur}$
3: **if** $diff$ is negative **then**
4: $diff \leftarrow -diff$
5: **end if**
6: Increase $e\_table_{agent}[s_x, s_y, a]$ by 1
7: Update UD with CMA, WA, or MRC variation
8: **end if**
9: Return UD

difference in past Q-value updates, which we refer to as the updateDifference variable, or simply UD. Note that UD is always a positive value because we take the absolute difference in Q-value changes when updating this value. For all 3 of the methods we present, UD is updated at most 2 times per time step. The first UD update occurs when the agent employing the method, A_0 in our case, takes a step in the environment. After updating its own policy $\pi_0 \in M$, A_0 will use the difference in Q-values it just updated, if any, for its own policy to update UD. A_0 can also use the policy data acquired after communication with other nearby agents to update UD, but only if the UD variable is uninitialized and has not been updated by A_0 previously (this is important for the case with multiple UD variables, as will be mentioned later). Algorithms 2 and 3 provide further details regarding these two methods of updating and using UD. Note that the second algorithm is calculated for each action $a \in A(s) \forall s \in S$ for policies $\{\pi_1, ..., \pi_{N-1}\} \in M$. These 2 algorithms have slight modifications depending on which of the 3 variations is used for updating UD.

In Algorithm 2, A_0 still updates its own π_0 policy with experience points the same way as in the original MEC method. Any Q-value change experienced by A_0 results in 1 experience point towards that (s,a) pair for π_0. The difference now is how A_0 evaluates and assigns experience points to policies $\{\pi_1, ..., \pi_{N-1}\} \in M$. A_0 defines experience points relative to the Q-value update differences it experiences or records.

CMA Method

The cumulative moving average (CMA) method, as shown in Equation (6.1), takes an average of the past n data points. As new datum points arrive, this method will add them and create a new cumulative average.

$$\mathrm{CMA}_{t+1} = \frac{x_{t+1} + t \cdot \mathrm{CMA}_t}{t+1} \text{ where } \mathrm{CMA}_0 = 0 \tag{6.1}$$

Algorithm 3 Updating UD During Communication Phase

Input: $UD, qValue_{new}, qValue_{cur}, s_x, s_y, a, agent$
Output: UD

if $qValue_{new} \neq qValue_{cur}$ **then**
    Initialize $expPoints \leftarrow 0$
    Initialize $diff = qValue_{new} - qValue_{cur}$
    if $diff$ is negative **then**
        $diff \leftarrow -diff$
    end if
    if $UD \neq 0$ **then**
        $expPoints \leftarrow \frac{diff}{UD}$
    else
        $UD \leftarrow diff$
    end if
    if $expPoints$ is 0 **then**
        $expPoints \leftarrow 1$
    end if
    Increase $e\_table_{agent}[s_x, s_y, a]$ by $expPoints$
end if
Return UD

For the purposes of this work, x_{t+1} is replaced with $diff$. This CMA variation for updating UD requires keeping track of the number of data points used in the cumulative average. Therefore, in addition to UD we also need to keep track of the updateCount variable, or UC. The CMA method is different from the other variations for updating UD because it considers all Q-value changes over the length of the run and averages them equally.

WA Method

The exponential, recency-weighted average, or simply, weighted average (WA) method, as shown in Equation (6.2), takes a weighted averaged of the past n data points and weights new datum points according to the value defined for α.

$$\text{WA}_{t+1} = \text{WA}_t(1 - \alpha) + \alpha(x_{t+1}) \text{ where } 0 < \alpha \leq 1 \tag{6.2}$$

Again, for the purposes of this work, x_{t+1} is replaced with $diff$. The WA method is different from the other variations for updating UD because it weights newer, or more recent, values with a different weight than it does the past averaged values.

MRC Method

The most recent change (MRC) method is the simplest of the 3 methods and updates UD by replacing it with the most recent value for recorded for $diff$, the absolute value of the difference between $qValue_{new}$ and $qValue_{cur}$.

6.4 Improving the MEC Method Results

6.4.1 Experiments

We now introduce the concepts of single-averaged value (SAV) and multiple-averaged value (MAV). As previously described, only one UD variable and possibly one UC variable was used to keep track of the average difference in Q-value updates across a simulation run. We denote this as an SAV algorithm. An MAV algorithm creates one UD variable and possibly one UC variable for each (s,a) pair in the Q-table. Therefore, if A_0 is in an environment with 64 states and 4 actions per state, A_0 would create a table to store 256 UD variables (and possibly 256 UC variables). The rationale for this being that after analyzing the Q-value updates amongst the (s,a) pairs, we noticed that certain areas of the environment experienced a greater range of Q-value updates. Therefore, combining the previous 3 MEC improvement algorithms with both the SAV and MAV version, we get a total of 6 methods to test in our experiments. For the WA method we test with $\alpha = 0.15$, 0.5, and 0.85. This represents a good range of values from 0 to 1. Additionally, when $\alpha = 0.15$ it takes on the same value as the α used for updating the RL algorithm, which may prove beneficial.

We conduct 2 main experiments in this section: evaluating the proposed MEC improvement algorithms against one another and comparing the one with the best learning performance against the original MEC method. For these experiments our simulation environment is the (8×8) scenario previously used.

Experiment 1: Evaluating the Proposed MEC Improvement Methods

In this section, we present the results of this experiment, as shown in Fig. 6.7 and Table 6.6. For this experiment, the communication range is set to 0. The learning performance of the original MEC method and the simulated fix to the MEC method are included in the graph as reference curves. According to the results, all proposed MEC improvement algorithms performed better than the original MEC method in this limited communication environment. The MAV variations of the methods, in general, performed better than the SAV variations, as expected. It was unexpected to receive results indicating that our methods performed better than the simulated fix to the problem. This indicates that our proposed improvement algorithms did better than the ideal version (no e-table distortion) of the original MEC

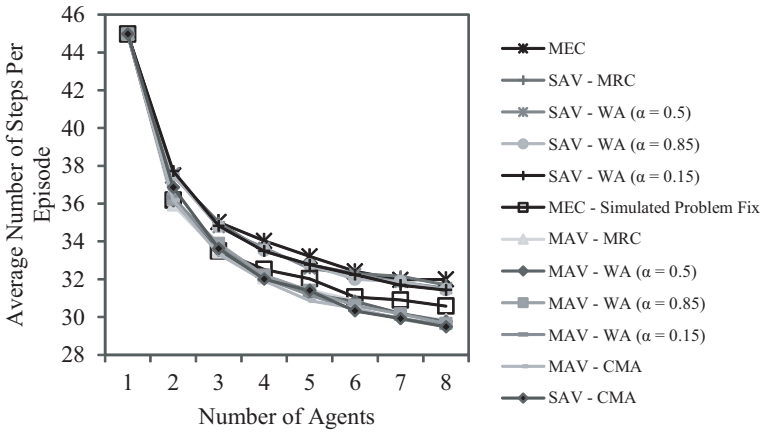


Figure 6.7: Evaluating Proposed MEC Improvement Algorithms.

Table 6.6: Evaluating Proposed MEC Improvement Algorithms.

Method	Average Steps
MEC	31.99 ± (0.22)
SAV - MRC	31.7 ± (0.23)
SAV - WA ($\alpha = 0.5$)	31.52 ± (0.22)
SAV - WA ($\alpha = 0.85$)	31.44 ± (0.21)
SAV - WA ($\alpha = 0.15$)	31.43 ± (0.22)
MEC - Simulated Problem Fix	30.58 ± (0.22)
MAV - MRC	29.84 ± (0.21)
MAV - WA ($\alpha = 0.5$)	29.76 ± (0.21)
MAV - WA ($\alpha = 0.85$)	29.66 ± (0.20)
MAV - WA ($\alpha = 0.15$)	29.55 ± (0.21)
MAV - CMA	29.49 ± (0.21)
SAV - CMA	29.48 ± (0.21)

method for environments with limited communication ranges.

Although the SAV – CMA method performed as well as the MAV – WA ($\alpha = 0.85$), MAV – WA ($\alpha = 0.15$), and MAV – CMA methods, the SAV – CMA method has a far smaller memory footprint than the other 3 methods because it only stores 1 copy of the UD variable and 1 copy of the UC variable. Therefore, for the next experiment we will use the SAV – CMA method to compare against the original MEC algorithm. The SAV – CMA method was the only SAV method to outperform the simulation fix to the problem. It is also interesting to note that the SAV – CMA and MAV – CMA methods performed equally well according to the 95% CI's calculated by a t-test.

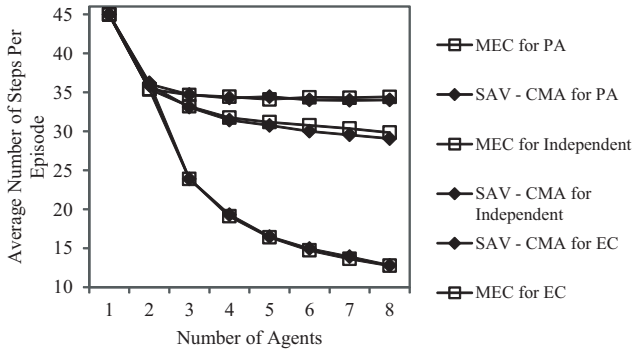


Figure 6.8: Comparison of Improved MEC Method to MEC Method.

Table 6.7: Comparison of Improved MEC Method to MEC Method.

A_0's Method	$\{A_1, ..., A_{N-1}\}$'s Method	Average Steps
MEC	PA	34.42 ± (0.26)
SAV - CMA	PA	34.02 ± (0.25)
MEC	Independent	29.84 ± (0.21)
SAV - CMA	Independent	29.06 ± (0.21)
SAV - CMA	EC	12.80 ± (0.12)
MEC	EC	12.75 ± (0.12)

Experiment 2: Comparison of Improved MEC Method to Original MEC Method

In this section, we present the results of the experiment comparing the SAV – CMA method to the original MEC method. To adequately test the improved MEC method, we test with A_0 employing the improved MEC method and $\{A_1, ..., A_{N-1}\}$ employing the Independent, PA, and EC methods. The results of this experiment are shown in Fig. 6.8 and Table 6.7. For this experiment, the communication range is set to ∞ to verify that the SAV – CMA method can also perform well in environments with no limit on communication.

According to Table 6.7, the differences in learning performance for both the Independent and PA tests are statistically significant. However, the differences in learning performance for the EC tests are not. This indicates that for 2 of the 3 tests, the SAV – CMA method outperforms the original MEC method and for the 3rd test it performs the same.

6.4.2 Summary

The SAV – CMA method performs well in both environments with limited communication ranges and environments with unrestricted communication. Considering that the SAV – CMA method significantly outperformed the original MEC method in a limited communication environment and performed as good as or better in unrestricted communication

environments, the SAV – CMA method is an overall improvement over the original MEC method in terms of learning performance.

Chapter 7

Scaling Environment Size

7.1 Scaling Environment Size Approach

It is of interest to understand how the proposed non-reciprocating methods' attributes scale when the simulation environment becomes larger. Specifically, we intend to determine how the learning performance evolves in addition to how the communication frequency, or bandwidth, consumed are affected.

7.2 Scaling Environment Size Results

7.2.1 Experiments

The maps we use in this experiment are blank maps that are of sizes (10 x 10), (12 x 12), (14 x 14), and (16 x 16). All agents start on the left, central side of the map and have a goal on the right, central side of the map, similar to our previous map. For this experiment we use the MEC method with a set communication range of 4.

To compare the learning performances and communication frequencies across the various map sizes, we perform averages for the first 118 episodes of the run. 118 episodes represents the number of episodes necessary for an independently learning agent to reach an average episode step count that converges within 1 step of the optimal step count for the (10 x 10) map.

Table 7.1: Necessary Episodes For Learning Performance Convergence for Increasing Map Size

Map Size	State Count	Episodes Necessary
10 × 10	100	118
12 × 12	144	185
14 × 14	196	240
16 × 16	256	320

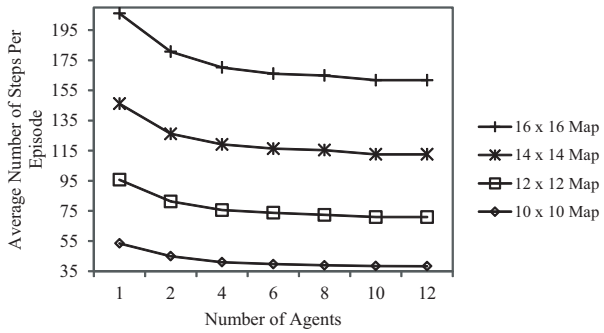
7.2.2 Results

Fig. 7.1a presents the results for learning performance as we scale the number of agents for the different map sizes. Larger maps correlate to a higher number of steps per episode for the agent. As more agents are added to the maps, the average number of steps per episode decreases. Increased map size also correlates to an increase in the differences in average number of steps per episode. This makes sense considering that an increase in map size equates to an increase of the difference between the number of states for the map. Fig. 7.1b presents the results for communication frequency as we scale the number of agents for the different map sizes. Larger maps correlate to a higher communication frequency among the agents. As more agents are added to the maps, the communication frequency increases. Increased map size correlates to a decrease in the differences in communication frequency.

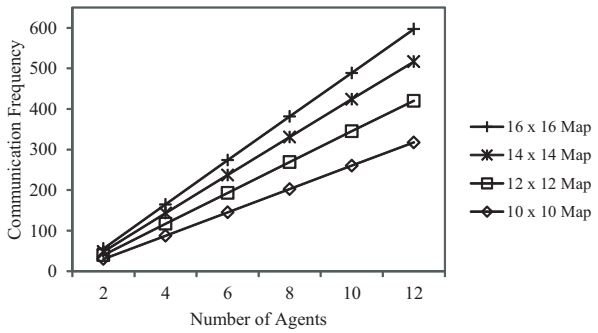
We provide Fig. 7.1c and Fig. 7.1d to reveal the average number of steps vs. number of episodes for the 4 maps with 1 and 12 agents. Fig. 7.1c is a close-up shot of Fig. 7.1d and shows the average number of steps for the first 60 episodes to provide better result clarity. These graphs provide further insight regarding the effects of scaling map size on learning performance. Namely, these graphs enable us to visualize the increasingly quicker initial learning phase as the maps scale up in size, followed by the increasingly longer period of converging to the optimal step count for the maps. Table 7.1 provides more information regarding precisely when the single agent curves reach step count convergence for each of the map sizes.

7.3 Large-Scale Hospital Simulation Approach

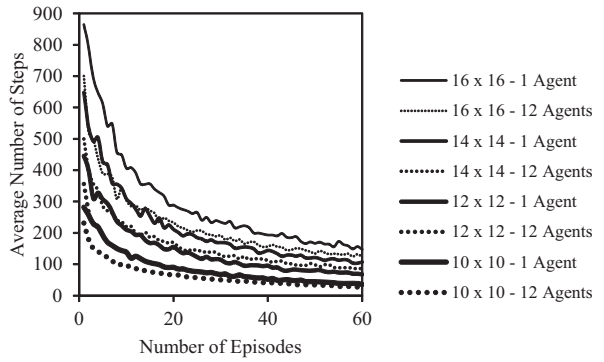
We present a large-scale simulation in a hospital-like environment to provide results for a visual comparison between the Independent learning method and the top performing non-reciprocating method, MEC with SAV – CMA. This simulation represents the culmination of this research.



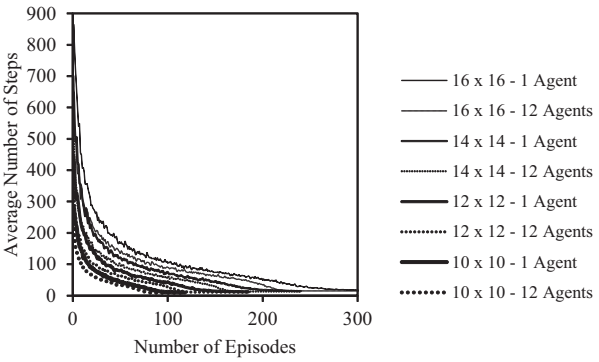
(a) Learning Performance for the MEC Method, Scaling Map Size.



(b) Communication Frequency for the MEC Method, Scaling Map Size.



(c) Learning Performance for MEC Method, Scaling Map Size.



(d) Learning Performance for MEC Method, Scaling Map Size.

Figure 7.1: Scaling Map Size Tests for the MEC Method.

7.4 Large-Scale Hospital Simulation Results

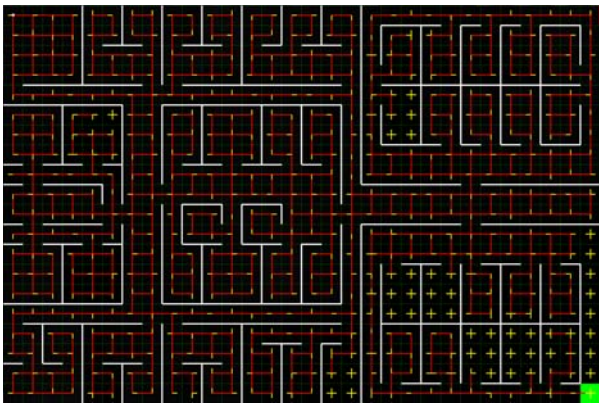
In this experiment we conduct 2 tests. The first test collects results from A_0 when it is using the Independent learning method in an environment with 7 other agents also using the Independent learning method. The second test collects results from A_0 when it is using the MEC with SAV – CMA learning method in an environment with 7 other agents using the Independent learning method. The hospital-like environment is simulated by using a map that represents a single floor of a hospital. This floor contains many hallways, large rooms, and small rooms. Agents start in the upper-left corner in a large room and are attempting to find the goal in the bottom-right corner at the end of a hallway. In this experiment the communication range is set to 6. Only 1 run, consisting of 500 episodes, is conducted.

We provide 3 forms of visuals in our results: heat map, agent trace, and agent policy. The heat map represents how often A_0 enters the various states in the environment. Each state s indicates how often A_0 has entered that state by using a grayscale color scheme. A darker shade represents low state visitation and a brighter shade represents high state visitation.

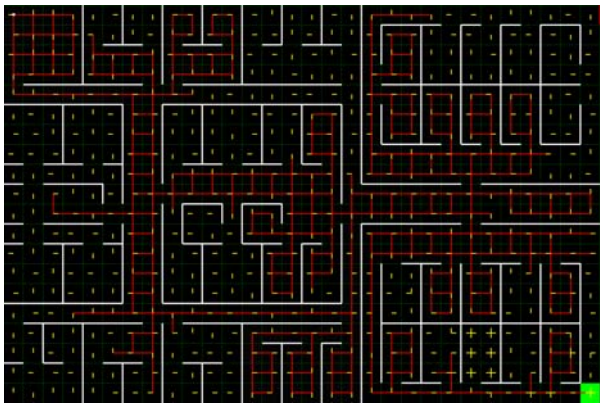
Note that the heat map is cumulative across the episodes. To make the heat map comparison relative between the 2 methods, the number used to divide the state count to calculate the grayscale shade was chosen as the highest accumulated state count from the two methods over the 500 runs (the value equalled 1726). The agent trace simply captures the route taken by A_0 during the episode. The route recorded does not indicate direction, but rather provides a general idea of the map coverage of A_0 during the episode. The agent trace is the red line in the figures. Finally, we also show the policy for A_0 $\forall s \in S$ for the current episode. The action a in state s with the highest Q-value is chosen as the best action to take given s. Ties result in multiple best actions chosen for the given state. The highest Q-value for each state s is represented by the yellow lines emanating from the center of the state. A yellow line extending from the center of the state and pointing North indicates that for that particular state, the North action has the highest Q-value.

The 3 types of visuals are overlaid for each of the 500 episodes. Fig. 7.2 provides screenshots from the simulation at trial numbers 0, 99, 199, 299, 399, and 499. Included in each of the subfigures is a count of the total number of steps taken during the episode.

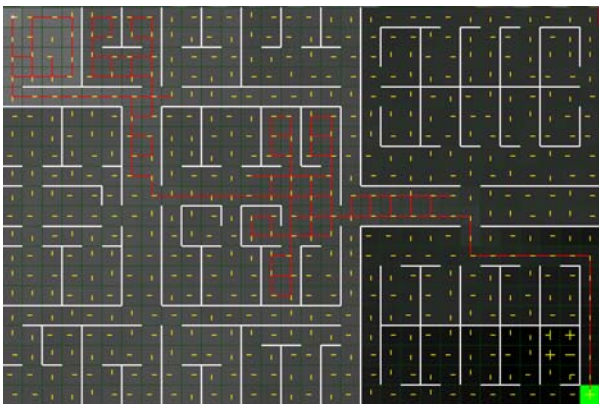
There are 2 related, reinforcing points to glean from the visuals in Fig. 7.2. The first point is that the MEC with SAV – CMA method converges more quickly to a near-optimal solution than the Independent learning method does. This can be seen by looking at a combination of the agent trace map coverage and the provided step count for each episode. The second point is that the MEC with SAV – CMA method exhibits a much lower heat map intensity across the environment. This is because the agent is able to utilize the policy information gathered from the other nearby agents, thereby reducing the number of states that need to be visited. The heat map for the MEC with SAV – CMA method is most intense around the near-optimal path whereas the heat map for the Independent method is more uniform in high state visitation across the entire map.



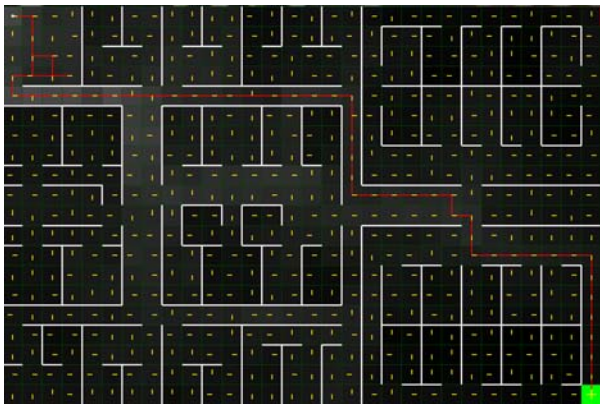
(a) Independent Method, Trial 0, 16236 Steps



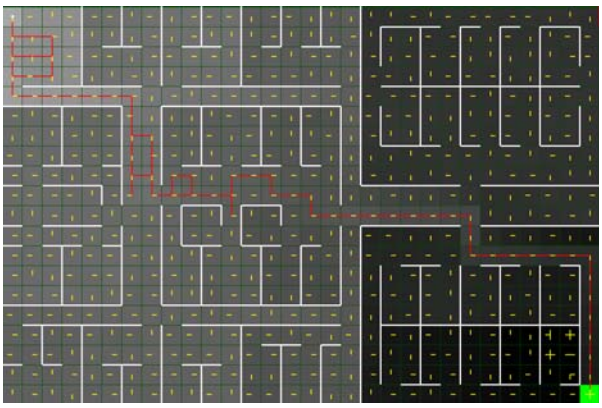
(b) MEC w/ SAV – CMA Method, Trial 0, 4203 Steps



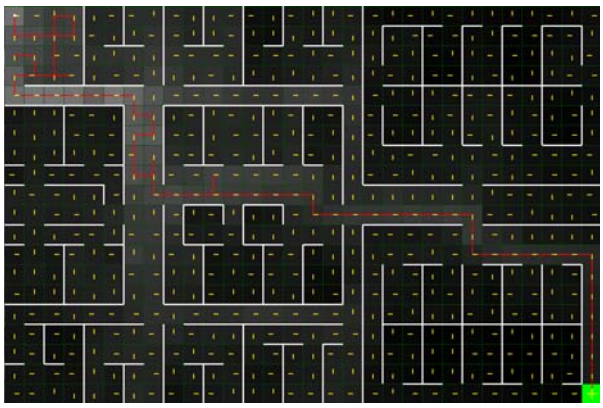
(c) Independent Method, Trial 99, 722 Steps



(d) MEC w/ SAV – CMA Method, Trial 99, 98 Steps

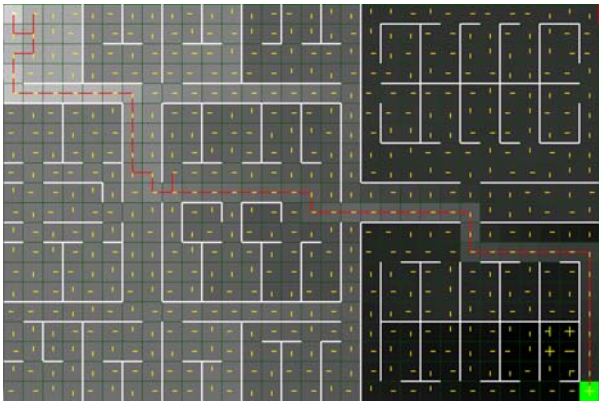


(e) Independent Method, Trial 199, 132 Steps

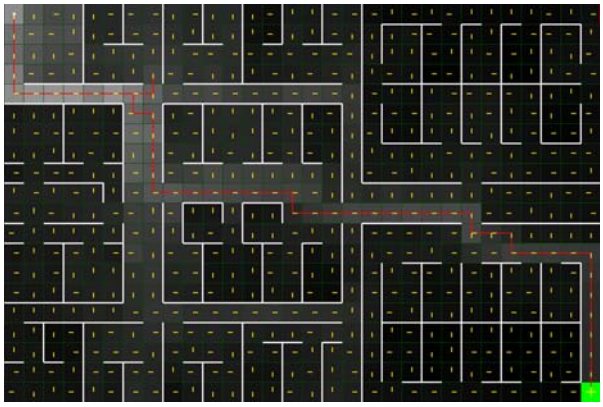


(f) MEC w/ SAV – CMA Method, Trial 199, 84 Steps

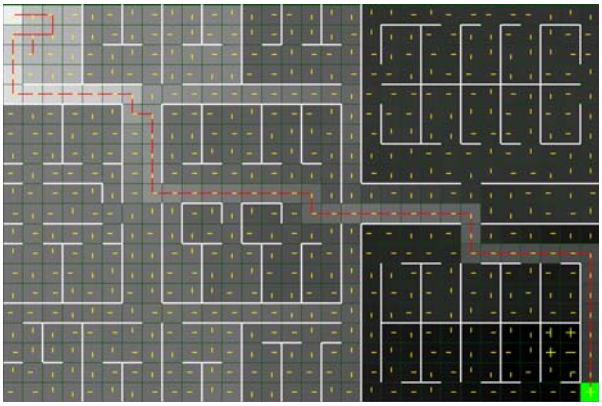
Figure 7.2: Heat Map, Agent Trace, and Agent Policy Visuals.



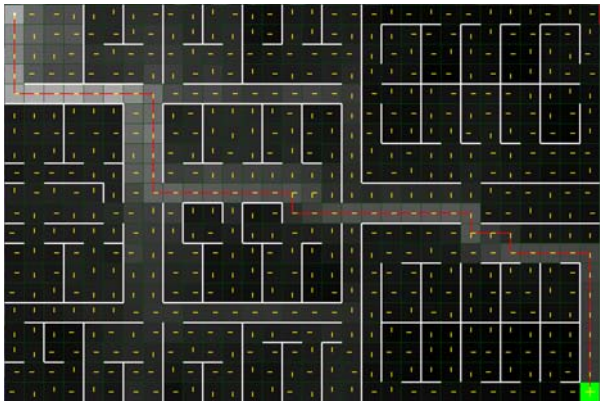
(g) Independent Method, Trial 299, 72 Steps



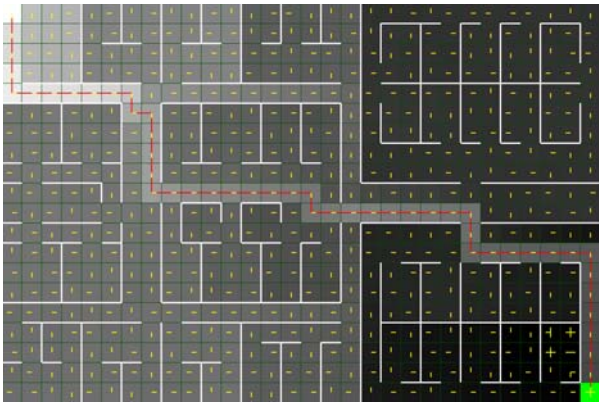
(h) MEC with SAV – CMA Method, Trial 299, 60 Steps



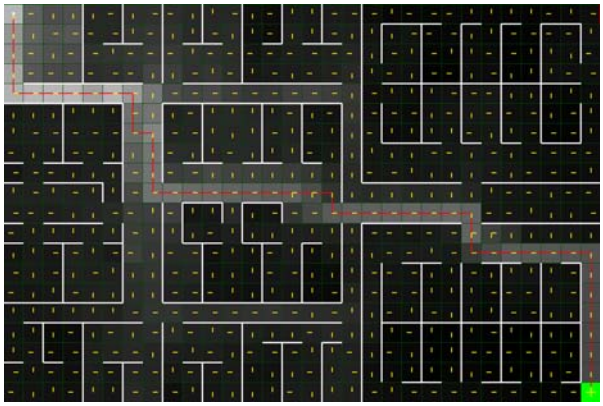
(i) Independent Method, Trial 399, 59 Steps



(j) MEC with SAV – CMA Method, Trial 399, 49 Steps



(k) Independent Method, Trial 499, 49 Steps



(l) MEC with SAV – CMA Method, Trial 499, 49 Steps

Chapter 8

Comparing Methods

In this chapter, we will briefly summarize and compare the memory costs associated with each of the following algorithms: pure SVS, HAV – Boltzmann, MA (version 2), MEC, HECA, and MEC with SAV – CMA. Additionally, we will also summarize and compare the algorithmic complexity of the methods. Finally, we conclude this chapter with an overall comparison using algorithmic complexity, memory usage, learning performance, and bandwidth consumption (for Type II learning) as metrics.

8.1 Pure SVS Method

The memory footprint associated with this method requires that the agent store N policies, $\{\pi_0, ..., \pi_{N-1}\} \in M$, where the size of each policy is $|S| \times |A|$. Compared to the original design for an agent employing standard Q-Learning, our method requires storing an additional $N-1$ policies. Technically, the pure SVS method could operate with only requiring memory to store approximately 1 policy and $N-1$ Q-values, however, in this work we assume that an agent communicating with another agent receives the entire Q-table with each access/request via communication.

The algorithmic complexity of the pure SVS method is unique from the other methods, excluding its derivative method, HAV – Boltzmann, in that it only adds additional work at phase 1. Referring back to Fig. 4.2, the diagram that visually depicts the 5 phases experienced by agents at each time step, we see that phase 1 is the action-selection phase. Whereas the other methods, again excluding the HAV – Boltzmann method, simply perform this phase by performing AS on π_0, the pure SVS method performs AS across $\{\pi_0, ..., \pi_{N-1}\}$ $\in M$ for state s. This results in more calculations for Equation 2.4 in order to determine an action to take given state s.

8.2 HAV – Boltzmann Method

The memory footprint associated with this method requires that the agent store N policies, $\{\pi_0, ..., \pi_{N-1}\} \in M$, where the size of each policy is $|S| \times |A|$. Additionally, the agent must store a policy representation value for each of the N policies.

The algorithmic complexity of this method scales better than the algorithmic complexity associated with the pure SVS method. This method makes use of policy representation values and therefore reduces the total number of actions to choose from for the first AS down to N, instead of the $4N$ actions that the pure SVS method would have to choose from. Following this, the agent would make a follow-up AS for 4 actions from the chosen policy. Similar to the pure SVS method, this method adds additional work exclusively at phase 1.

8.3 MA (Version 2) Method

The memory footprint associated with this method requires that the agent store $N + 1$ policies, or Q-tables, $\{\pi_0, ..., \pi_N\} \in M$, where the size of each policy is $|S| \times |A|$. Compared to the reciprocating PA method, our MA V2 method requires memory for storing N additional policies. The MA V2 method could operate with only requiring memory to store 2 policies, however, as previously mentioned, in this work we assume that an agent communicating with another agent receives the entire Q-table per communication request.

The algorithmic complexity of the MA V2 method is more straightforward compared to the other methods. According to Fig. 4.2, the MA V2 method only operates at the levels of phases 4 and 5. In phases 4 and 5, the method simply 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 and assigns the averaged Q-value for each (s,a) pair to the corresponding entry in π_N.

8.4 MEC Method

The memory footprint associated with this method requires that the agent store N policies, $\{\pi_0, ..., \pi_{N-1}\} \in M$, where the size of each policy is $|S| \times |A|$. Additionally, the agent must store N E-tables, where the size of each E-table is $|S| \times |A|$. Compared to the reciprocating EC method, our MEC method requires memory for storing an additional $N - 1$ policies and $N - 1$ E-tables.

The algorithmic complexity of the MEC method is more complicated than that of the MA V2 method's algorithmic complexity. According to Fig. 4.2, the MEC method operates at the levels of phases 2, 3, 4, and 5. In phase 2, the agent updates π_0 based on the experienced

gained by taking action a in state s. Also in this phase, the MEC method must update the E-table for π_0 according to the direct experience recently acquired. In phase 3, the MEC method accesses the other agents' Q-tables and replaces $\{\pi_1, ..., \pi_{N-1}\}$ with the updated values. Additionally, in phase 3 while the old Q-tables are being replaced by the new Q-tables, the MEC method will update the E-tables according to any change in Q-values. In phases 4 and 5, the MEC method compares the E-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 and assigns the Q-value corresponding to the highest E-value (ties are broken randomly) for each (s,a) pair to the corresponding entry in π_0.

8.5 HECA Method

The memory footprint associated with this method is the same as the MEC method's memory footprint. The algorithmic complexity of the HECA method is the same as the MEC method's algorithmic complexity, with the exception that the HECA method resolves E-value ties by averaging the associated Q-values instead of randomly picking.

8.6 MEC with SAV – CMA Method

The memory footprint associated with this method is the same as the MEC method's memory footprint, with the exception that 2 additional variables, UD and UC, must also be stored. The algorithmic complexity of the MEC with SAV – CMA method is also identical to that of the MEC method's algorithmic complexity, with 2 differences. The first difference is that in phase 2, this method also updates the UD and UC variables. The second difference is that in phase 3, instead of detecting any change in Q-value and assigning an experience point to (s,a), this method will perform more steps and use the UD value to evaluate and assign experience points to (s,a) for each of $\{\pi_1, ..., \pi_{N-1}\}$.

8.7 Summary

In Table 8.1 we present a summary of the results comparing the 6 methods against one another with algorithmic complexity, memory usage, learning performance, and bandwidth consumption (for Type II learning) as metrics. The numbers in the table represent the relative rankings of the algorithms with respect to one another for the metric being used, with 1 being the highest and 4 being the lowest (accounting for ties).

We note that these rankings are meant to provide a general sense of how the algorithms compare to one another with regard to the metrics and the implementations used in this

Table 8.1: Comparing Algorithms. A '*' Denotes the Metric Winner.

Metric	Pure SVS	HAV – Boltzmann	MA V2	MEC	HECA	SAV – CMA
Learning Performance (Best)	3	3	3	2	2	1*
Bandwidth Consumption (Lowest)	3	3	3	2	2	1*
Memory Usage (Lowest)	1*	2	2	3	3	3
Algorithmic Complexity (Lowest)	1*	1*	2	3	3	4

research. They are not meant to serve as strict truths. Due to various factors like the number of states, the number of actions per state, the number of agents, algorithm implementation, agent structures with possible compression measures, etc., the relative rankings are subject to change.

As has been made clear previously in the work, the MEC with SAV – CMA method has the best learning performance rates of all 6 methods. This method's learning performance is closely followed by both the MEC and HECA methods' learning performance and distantly followed by the the pure SVS, HAV – Boltzmann, and MA V2 methods' learning performances.

For the bandwidth consumption metric, we are considering bandwidth consumption for Type II learning because it more accurately assesses this metric when compared to Type I learning. The bandwidth consumption metric aligns with the learning performance metric, as an algorithm that performs more quickly will generally exit the environment more quickly and communicate with other agents less; the exception being when an environment has a severely limited communication range, as aforementioned.

Regarding memory usage, the MEC with SAV – CMA method requires the most memory of all 6 methods. It is followed by the MEC and HECA methods, which tie, and by the MA V2 and HAV – Boltzmann methods. Finally, the pure SVS method has the smallest memory footprint.

According to Big-O notation, all 6 methods are equal in terms of algorithmic complexity. However, as Big-O notation only describes an upper bound on algorithm growth rate, we analyzed the methods in greater detail above. The pure SVS and HAV – Boltzmann methods are associated with the lowest algorithmic complexity, followed by the MA V2 method. Following these 3 methods are the MEC and HECA methods. Note that their algorithmic complexity is too close to necessitate differentiation in terms of ranking. Finally, the MEC with SAV – CMA method is ranked last and is associated with the highest algorithmic complexity.

Chapter 9

Conclusions and Future Work

Given a Q-table from another homogeneous, cooperative agent, what is the best method to use to extract salient data to improve learning performance? Results from this work indicate that while the MEC with SAV – CMA method performs best overall in terms of learning performance for both limited and unlimited communication environments, it also requires the most memory and is the most algorithmically complex of the proposed methods. Therefore, choosing the best of the proposed methods to use depends on the constraints of both the physically embodied agent, or robot, and the environment in which it will exist. Results from the experiments show that it is possible for agents to improve learning performance, when compared to the Independent method, in environments where other agents do not reciprocate with it.

This research opens up the possibility for future work. In this work we focused on improving learning performance as the prime objective and analyze, but do not improve upon, metrics such as communication bandwidth, memory usage, and algorithmic complexity. We assume that the communication bandwidth between A_0 and the other agents strictly consists of transferring a Q-table, whereas in reality, this could be greatly improved by utilizing methods that transfer only specific Q-values. This would present an interesting challenge as agents would have to devise some form of evaluation method to determine which Q-values to intelligently query. Along these lines, methods could be developed to throttle the frequency of contact with another agent when bandwidth is limited to keep communication frequency lower. Tan mentions that another method for keeping communication frequency down is to only contact other agents when the agent does not have confidence in actions for specific states [50]. Additionally, agents could share episodal information in the form of (sensation, action, reward) instead of policies or specific Q-values. This type of sharing is mentioned in Tan's work [50]. Tan's work also discusses how the receiving agent 'replays' the episode to assimilate the knowledge into its own policy. This could be a possible avenue to explore for cases when the exposed agent interface is something other than Q-tables.

We 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 are physically embodied agents, this assumption of a large memory may prove to be unrealistic. Suppose that the number of agents in the environment is large and/or that the state space is large. If an agent cannot store the entire Q-tables of the other agents, how could our methods be modified to adapt to this type of a situation? Newer methods use function approximation to reduce the state space, how could our methods be adapted to work in such environments? Future work could study how our methods scale and adapt to continuous-environment domains [44]. We assume that the algorithms operate in terms of time steps and do not consider the actual wall-clock time that may be associated with the proposed methods. Future work can study how the methods proposed in this paper can be adapted to perform given such constraints.

A list of miscellaneous, additional future work is as follows:

- How well do our methods perform when agents have noisy input?
- How well do our methods adapt when other methods of learning (like SARSA, Actor-Critic, etc.) are employed instead of Q-learning?
- We assume a reliable communication channel among the agents where Q-tables queries are always satisfied within communication range and the information transferred contains no errors. How would these modifications to the simulation scenario affect the proposed methods?

Research into areas such as these could greatly improve the applicability and usefulness of our work.

Bibliography

[1] Osman Abul, Faruk Polat, and Reda Alhajj. Multiagent reinforcement learning using function approximation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 30(4):485–497, 2000.

[2] Sorin Achim, Peter Stone, and Manuela Veloso. Building a dedicated robotic soccer system. In *Proceedings of the IROS-96 Workshop on RoboCup*, pages 41–48, Osaka, Japan, November 1996.

[3] Majid Nili Ahmadabadi and Masoud Asadpour. Expertness based cooperative q-learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 32(1):66–76, 2002.

[4] Majid Nili Ahmadabadi, Masoud Asadpour, and Eiji Nakano. Cooperative q-learning: the knowledge sharing issue. *Advanced Robotics*, 15(8):815–832, 2001.

[5] Majid Nili Ahmadabadi, Ahmad Imanipour, Babak Nadjar Araabi, Masoud Asadpour, and Roland Siegwart. Knowledge-based extraction of area of expertise for cooperation in learning. In *IROS*, pages 3700–3705. IEEE, 2006.

[6] Babak Nadjar Araabi, S. Mastoureshgh, and Majid Nili Ahmadabadi. A study on expertise of agents and its effects on cooperative q-learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(2):398–409, 2007.

[7] Tucker Balch. Reward and diversity in multirobot foraging. In *In IJCAI-99 Workshop on Agents Learning About, From and With Other Agents*, 1999.

[8] Samuel Barrett, Katie Genter, Todd Hester, Piyush Khandelwal, Michael Quinlan, Peter Stone, and Mohan Sridharan. Austin Villa 2011: Sharing is caring: Better awareness through information sharing. Technical Report UT-AI-TR-12-01, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, January 2012.

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

[10] Michael Bowling. *Multiagent learning in the presence of agents with limitations*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2003. AAI3121053.

[11] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[12] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(2):156–172, 2008.

[13] David Carmel and Shaul Markovitch. Opponent modeling in a multi-agent system. In *Lecture note in AI, 1042: Adaptation and Learning in Multi-agent Systems, Lecture Notes in Artificial Intelligence*, pages 40–52. Springer-Verlag, 1995.

[14] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In Jack Mostow and Chuck Rich, editors, *AAAI/IAAI*, pages 746–752. AAAI Press / The MIT Press, 1998.

[15] J. Clouse. On integrating apprentice learning and reinforcement learning. Technical report, University of Massachusetts, Amherst, MA, USA, 1997.

[16] Valentino Crespi, George Cybenko, Daniela Rus, and Massimo Santini. Decentralized control for coordinated flow of multi-agent systems, 2002.

[17] Bryan Cunningham and Yong Cao. Levels of realism for cooperative multi-agent reinforcement learning. In Ying Tan, Yuhui Shi, and Zhen Ji, editors, *ICSI (1)*, volume 7331 of *Lecture Notes in Computer Science*, pages 573–582. Springer, 2012.

[18] Bryan Cunningham and Yong Cao. Non-reciprocating sharing methods in cooperative q-learning environments. In *IAT*. (In Press), 2012.

[19] Jrg Denzinger and Matthias Fuchs. Experiments in learning prototypical situations for variants of the pursuit game. In *In Proceedings on the International Conference on Multi-Agent Systems (ICMAS-1996*, pages 48–55. MIT Press, 1995.

[20] F. Fernández and L.E. Parker. Learning in large cooperative multi-robot domains, 2001.

[21] Fernando Fernández, Javier García, and Manuela M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, 2010.

[22] Fernando Fernández and Manuela M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *AAMAS*, pages 720–727. ACM, 2006.

[23] Amy Greenwald, Martin Zinkevich, and Pack Kaelbling. Correlated q-learning. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 242–249, 2003.

[24] Carlos Guestrin Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *In Proceedings of the ICML-2002 The Nineteenth International Conference on Machine Learning*, pages 227–234, 2002.

[25] Yu han Chang, Tracey Ho, and Leslie Pack Kaelbling. All learning is local: Multi-agent learning in global reward games.

[26] Junling Hu and Michael P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm, 1998.

[27] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[28] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: extracting reusable task knowledge from visual observation of human performance. *Robotics and Automation, IEEE Transactions on*, 10(6):799 –822, dec 1994.

[29] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann, 2000.

[30] Laurens Leerink, Simon R. Schultz, and Marwan A. Jabri. A reinforcement learning exploration strategy based on ant foraging mechanisms, 1995.

[31] Francisco Martinez-Gil, Fernando Barber, Miguel Lozano, Francisco Grimaldo, and Fernando Fernández. A reinforcement learning approach for multiagent navigation. In Joaquim Filipe, Ana L. N. Fred, and Bernadette Sharp, editors, *ICAART (1)*, pages 607–610. INSTICC Press, 2010.

[32] Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández. Multi-agent reinforcement learning for simulating pedestrian navigation. In Peter Vrancx, Matthew Knudson, and Marek Grzes, editors, *ALA*, volume 7113 of *Lecture Notes in Computer Science*, pages 54–69. Springer, 2011.

[33] N. D. Monekosso, Paolo Remagnino, and Adam Szarowicz. An improved q-learning algorithm using synthetic pheromones. In *Revised Papers from the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems: From Theory to Practice in Multi-Agent Systems*, CEEMAS '01, pages 197–206, London, UK, UK, 2002. Springer-Verlag.

[34] Ndedi Monekosso and Paolo Remagnino. Phe-q : A pheromone based q-learning. In Markus Stumptner, Dan Corbett, and Mike Brooks, editors, *AI 2001: Advances in Artificial Intelligence*, pages 345–356. Springer Berlin / Heidelberg, 2001.

[35] Ndedi Monekosso and Paolo Remagnino. An analysis of the pheromone q-learning algorithm. In Francisco Garijo, Jos Riquelme, and Miguel Toro, editors, *Advances in Artificial Intelligence IBERAMIA 2002*, pages 224–232. Springer Berlin / Heidelberg, 2002.

[36] Bilge Mutlu and Jodi Forlizzi. Robots in organizations: the role of workflow, social, and environmental factors in human-robot interaction. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, HRI '08, pages 287–294, New York, NY, USA, 2008. ACM.

[37] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *NIPS*. MIT Press, 2003.

[38] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[39] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *CoRR*, abs/1106.4569, 2011.

[40] Matt Quinn. Evolving communication without dedicated communication channels. In Jozef Kelemen and Petr Sosk, editors, *Advances in Artificial Life*, pages 357–366. Springer Berlin / Heidelberg, 2001.

[41] Richardson Ribeiro, André Pinz Borges, and Fabrício Enembreck. Interaction models for multiagent reinforcement learning. In Masoud Mohammadian, editor, *CIMCA/IAWTIC/ISE*, pages 464–469. IEEE Computer Society, 2008.

[42] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[43] Peter Stone. *Intelligent Autonomous Robotics: A Robot Soccer Case Study*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2007.

[44] Peter Stone, Gregory Kuhlmann, Matthew E. Taylor, and Yaxin Liu. Keepaway soccer: From machine learning testbed to benchmark. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *RoboCup*, volume 4020 of *Lecture Notes in Computer Science*, pages 93–105. Springer, 2005.

[45] Peter Stone and Manuela Veloso. Layered learning. In Ramon López de Mántaras and Enric Plaza, editors, *Machine Learning: ECML 2000 (Proceedings of the Eleventh European Conference on Machine Learning)*, pages 369–381. Springer Verlag, Barcelona,Catalonia,Spain, May/June 2000.

[46] Peter Stone and Manuela M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Auton. Robots*, 8(3):345–383, 2000.

[47] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[48] Y. Takahashi and M. Asada. Behavior acquisition by multi-layered reinforcement learning. In *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, volume 6, pages 716 –721 vol.6, 1999.

[49] Y. Takahashi and M. Asada. Multi-controller fusion in multi-layered reinforcement learning. In *Multisensor Fusion and Integration for Intelligent Systems, 2001. MFI 2001. International Conference on*, pages 7 – 12, 2001.

[50] Ming Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In *ICML*, pages 330–337. Morgan Kaufmann, 1993.

[51] P. Tangamchit, J.M. Dolan, and P.K. Khosla. The necessity of average rewards in cooperative multirobot learning. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 2, pages 1296 – 1301 vol.2, 2002.

[52] Lisa Torrey. Crowd simulation via multi-agent reinforcement learning. In G. Michael Youngblood and Vadim Bulitko, editors, *AIIDE*. The AAAI Press, 2010.

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

[54] Barry Brian Werger and Maja J Matari'c. Exploiting embodiment in multi-robot teams. Technical report, University of Southern California, 1999.

[55] Ashley Wharton. Simulation and investigation of multi-agent reinforcement learning for building evacuation scenarios. Master's thesis, St Catherine's College, Oxford, Oxfordshire, UK, 2009.

[56] Marco Wiering, Rafal Salustowicz, and Jürgen Schmidhuber. Reinforcement learning soccer teams with incomplete world models. *Auton. Robots*, 7(1):77–88, 1999.

[57] T. Yamaguchi, Y. Tanaka, and M. Yachida. Speed up reinforcement learning between two agents with adaptive mimetism. In *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 2, pages 594 –600 vol.2, sep 1997.

[58] Yongming Yang, Yantao Tian, and Hao Mei. Cooperative q learning based on blackboard architecture. In *Proceedings of the 2007 International Conference on Computational Intelligence and Security Workshops*, CISW '07, pages 224–227, Washington, DC, USA, 2007. IEEE Computer Society.

[59] Lavi M. Zamstein. Koolio: Path planning using reinforcement learning on a real robot platform. In *University of Florida*, pages 25–26, 2006.

[60] Ping Zhang, Xiujun Ma, Zijian Pan, Xiong Li, and Kunqing Xie. Multi-agent cooperative reinforcement learning in 3d virtual world. In Ying Tan, Yuhui Shi, and Kay Chen Tan, editors, *ICSI (1)*, volume 6145 of *Lecture Notes in Computer Science*, pages 731–739. Springer, 2010.

Appendix A

Acronyms

Table A.1: Acronyms and their Descriptions.

Acronym	Description
AAV	average action value
AS	action-selection
CMA	cumulative moving average
EC	experience counting
HAV	highest action value
HECA	hybrid experience counting and averaging
IL	independent learner
JAL	joint action learner
MA	modified averaging
MARL	multi-agent reinforcement learning
MAS	multi-agent system
MAV	multiple-averaged value
MDP	Markov decision process
MEC	modified experience counting
MRC	most recent change
PA	policy averaging
PPR	probabilistic policy reuse
RL	reinforcement learning
SAV	single-averaged value
SVS	self vs. shared
UC	update count
UD	update difference
WA	weighted average
WSS	weighted-strategy sharing