The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

HogRider: Champion Agent of Microsoft Malmo Collaborative AI Challenge

Yanhai Xiong,* Haipeng Chen,*Mengchen Zhao, Bo An
Nanyang Technological University
{yxiong003,chen0939,zhao0204,boan}@ntu.edu.sg

Abstract

It has been an open challenge for self-interested agents to make optimal sequential decisions in complex multiagent systems, where agents might achieve higher utility via collaboration. The Microsoft Malmo Collaborative AI Challenge (MCAC), which is designed to encourage research relating to various problems in Collaborative AI, takes the form of a Minecraft mini-game where players might work together to catch a pig or deviate from cooperation, for pursuing high scores to win the challenge. Various characteristics, such as complex interactions among agents, uncertainties, sequential decision making and limited learning trials all make it extremely challenging to find effective strategies. We present HogRider - the champion agent of MCAC in 2017 out of 81 teams from 26 countries. One key innovation of HogRider is a generalized agent type hypothesis framework to identify the behavior model of the other agents, which is demonstrated to be robust to observation uncertainty. On top of that, a second key innovation is a novel Q-learning approach to learn effective policies against each type of the collaborating agents. Various ideas are proposed to adapt traditional Q-learning to handle complexities in the challenge, including state-action abstraction to reduce problem scale, a warm start approach using human reasoning for addressing limited learning trials, and an active greedy strategy to balance exploitation-exploration. Challenge results show that HogRider outperforms all the other teams by a significant edge, in terms of both optimality and stability.

Introduction

In complex multiagent systems, sequential decision making has long been a significant challenge with many characteristics. One prominent characteristic is the *interactions* among the agents. In many practical scenarios (e.g., the Stag-Hunt game), agents need to *cooperate* with each other to achieve a common goal with high rewards (e.g., hunting stags), while each individual agent is *self-interested* and might deviate from the cooperation for less risky goals with low rewards (e.g., hunting rabbits). Another critical characteristic comes from various *uncertainties*. One type of uncertainty arises from the lack of accurate knowledge of environment and other agents where the uncertain information can be modelled probabilistically. A more challenging type of uncertainty lies in the environment-related factors which we do not know how to model.

These two challenges are significantly amplified when it comes to *sequential decision making*, where we need to look at not only short term rewards but also rewards in the long run. Therefore, one has to consider subsequent effect of the current action, especially in a dynamic environment. Another crucial characteristic is the *limited learning trials*. On the Minecraft platform, it usually takes several seconds to complete one episode of game. Therefore, it is extremely time consuming to learn an effective policy.

The Microsoft Malmo Collaborative AI Challenge (MCAC), which is designed to encourage research relating to various problems in Collaborative AI, builds on a Minecraft mini-game (Mojang 2017) called "Pig Chase". Pig Chase is played on a 9×9 grid where agents can either work together to catch the pig and achieve high scores, or give up cooperation and achieve low scores. After playing certain episodes (e.g., 100) of games, the agent who achieves the highest average scores wins the challenge. Despite its simple rules, this game has all the above stated key characteristics. Though there are numerous papers studying sequential decision making in complex multiagent systems, they only address a subset of these characteristics. We hope to shed some light on solving this class of problems by presenting HogRider, a champion agent which won 2017 Microsoft Malmo Collaborative AI Challenge (Kuno and Schwiderski-Grosche 2017), with the following key contributions.

First, via enormous simulations, we provide thorough analysis of the Minecraft game, and identify its key challenges. Specifically, we discover several important aspects of the game that are not revealed by the MCAC rule, such as the pig's uncertain behavior model, the uncertainty in observing the other agent's actions, among others.

Second, we propose a novel agent type hypothesis approach for dealing uncertainties about the type of the other agent and the observation of the other agent's actions. When forming ad hoc teams, it is important to identify the types of the collaborating agents. A typical approach is to maintain a belief of their type, and update it based on the actions of the other agents using Bayes theorem (Barrett, Stone, and

*Denotes equal contribution.

Kraus 2011). However, we discover that the actions of the other agents could be incorrectly observed. To this end, we derive two adaptations to the traditional Bayes theorem to cope with this observation uncertainty.

Third, we come up with a novel Q-learning framework to learn an optimal policy against each type of agent, with the following novelties. First, state-action abstraction is utilized to reduce the huge state-action space. Besides, in classical tabular Q-learning, Q-values are usually randomly initialized, which requires a particularly large number of episodes for training. While it takes several seconds to complete an episode on the Minecraft platform, it is extremely time consuming to train an effective policy. We propose a warm start approach to initialize the tabular Q-function, which utilizes a decision tree framework derived from human reasoning. Moreover, we show that when learning trials are limited, random exploration of potential optimal actions is inefficient and sometimes even harmful for searching optimal policies. Therefore, we propose an active-ε-greedy method to make better tradeoff between exploitation and exploration.

Last, we conduct extensive experiments to evaluate HogRider. We demonstrate that by considering uncertainty in agent action observation, HogRider outperforms existing agent type belief update methods. We also show improved performance of HogRider compared with traditional Q-learning methods. Challenge results show that HogRider outperforms all the other teams by a significant edge, in terms of both optimality and stability. Moreover, we show that HogRider performs even better than human players.

Related Work

One line of related research is teamwork within a multiagent system where agents cooperate through communication (Tambe 1997; Grosz and Kraus 1996) or precoordination (Horling et al. 1999). These methods assume fully cooperative agents and do not consider uncertainty about cooperating agents' types. Recently, research efforts have been put on ad hoc interactions among agents, where agents need to interact without prior knowledge of the other agents (Stone et al. 2010). They study both cooperative (Barrett et al. 2013; Albrecht, Crandall, and Ramamoorthy 2016) and competitive (Southey et al. 2012; Albrecht, Crandall, and Ramamoorthy 2016) scenarios, where agent type hypothesis and opponent modelling are used to handle the uncertainty. However, they do not consider the uncertainty in agent action observation and limited learning trials.

Multi-agent reinforcement learning (MARL) has also been applied to study interactions among a set of agents (Singh, Kearns, and Mansour 2000; Conitzer and Sandholm 2007; Foerster et al. 2016). These approaches, from dealing with small to large scale problems, usually focus on obtaining Nash equilibrium via self-play, or learning optimal strategies against stationary opponents.

Another line of research studies the sequential multiagent planning problem via game theoretic approaches (Dunne et al. 2010; De Cote et al. 2012). Brafman et al. (2009) apply game theoretic solutions to coalition planning games and auction planning games. Jordán and Onaindia (2015) propose a two-game proposal for solving congestion games among a group of autonomous vehicles (Jordán et al. 2017).

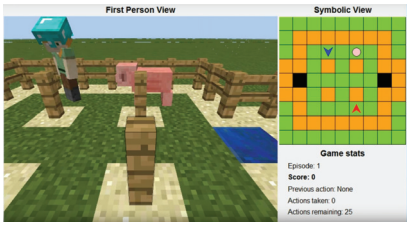


Figure 1: Setting of MCAC. The left hand side of the figure is the first person view of the game, while the right hand side of the figure is the symbolic global view.

Some other papers (Sandholm 2015; Horák, Bosanskỳ, and Pechoucek 2017) apply a stochastic game framework to multiagent planning problems. Despite the theoretical merits, pure game theoretic approaches usually suffer from computational issues that arise from large-scale state and strategy spaces. Other classical approaches to sequential planning in multiagent systems include decentralized partially observable Markov decision process and its variants (Seuken and Zilberstein 2008; Agmon and Stone 2012). These approaches also suffer from high computational complexity.

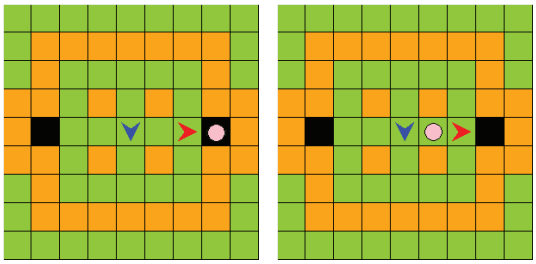
Challenge Description and Analysis

We briefly introduce the challenge background and analyze its key characteristics.

Settings & Rules

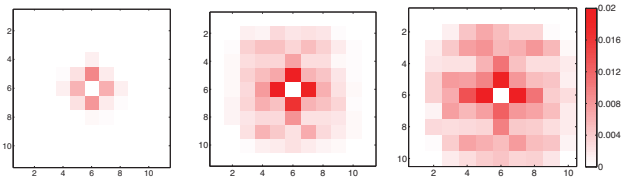
MCAC builds on a mini-game played within a 9×9 grid on the Minecraft platform (Figure 1). The green grids represent the grass where agents can walk on, the yellow grids represent the blocks where agents cannot walk on or pass through, and the two black grids represent two exits. There are two agents and a pig (which can be viewed as part of the environment). The red arrow represents the agent (RED) that we are going to design, the blue arrow represents the agent (BLUE) that agent RED needs to play with, and the pink dot represents a pig which needs to be caught. The *legal movements* of the agents include: move forward (*MF*), turn left (*TL*) and turn right (*TR*). Agent BLUE is provided by MCAC, which is designed to be either a *Random* agent or a *Focused* agent in each episode of game. A *Random* agent takes any of the three legal movements with an equal probability of $1/3$, while a *Focused* agent always chases the pig through the shortest path using A* search algorithm. In each episode of game, there is a probability of .25 for agent BLUE to be *Random*, and a probability of .75 to be *Focused*. This type does not change within one episode of game.

The goal of the challenge is to achieve a high score in certain episodes (e.g., 100 or 500) of games. After an episode started, two agents take actions in turn. If the pig is caught, each agent receives 25 points. To catch a pig, the pig has to be completely surrounded by blocks and the agents. Figure 2(a) shows an example where a pig can be caught by a single agent, i.e., when the pig goes to the exit, and one



(a) Catch a pig alone (b) Catch a pig together

Figure 2: Two types of situations where a pig is caught



(a) (b) (c)

Figure 3: Probability distribution of the pig's positions after agent RED takes one step, where the grid in the center is the original position of the pig, and darker color means higher probability. In Figures 3(a)-3(c), agent RED respectively waits for (a) no time (b) 1 second, and (c) 3 seconds before it takes a step. Note the pig does have a very short time (around 0.1 seconds) moving even if agent RED does not wait. This time is the response time of the Minecraft platform.

agent goes to the only grass grid beside the exit. Figure 2(b) shows an example where a pig can only be caught by two agents. An alternative is to give up cooperation and go to the exit. In this case, only the agent who *first* reaches the exit can get 5 points. Our agent loses 1 point whenever one step is made, and one episode of game terminates as long as one of the following conditions is met: 1) the pig is caught, 2) one agent reaches the exit, 3) 25 steps are taken, or 4) the maximum game time (about 100 seconds) is used up.

Analysis and Key Challenges

Uncertain behavior model of the pig. The behavior model of the pig is crucial to generating efficient policies, whereas it is not released by MCAC. We keep statistics of the pig's positions after our agent takes one step. After $10,000$ steps, we draw the probability distribution of the pig's positions. As shown in Figure 3, we can see that 1) the pig does not follow the legal movements as human agents, and is able to move multiple steps or even make turns when human agent takes one step, 2) the probability of the pig moving to each direction is almost equal, and 3) with longer waiting time before our agent RED takes a step, the higher probability the pig moves. The statistics gives us a hint that it might be beneficial to wait for a few seconds when the pig is in a position that is uncatchable, so that it might move to positions that are catchable.

Uncertainties in the type of agent BLUE & observation of its actions. In each episode of game, agent BLUE has a .25 and a .75 probability of being *Random* or *Focused*, respectively. While correctly predicting its type relies on accurate observation of its movements, we noticed that the action observed (returned by the Minecraft simulator) has a probability of around .25 to be incorrect. It would be highly risky if this observation uncertainty is neglected.

Coexistence of cooperation & self-interest. To catch a pig (25 points) in situations like in Figure 2(b), agent RED must cooperate with agent BLUE. Under some other situations, it might be more beneficial to deviate from the cooperation and move to the exit (5 points), especially when agent RED is close to the door, or there are few steps left.

Sequential decision making. Apart from the above challenges, our agent needs to sequentially make decisions during each episode of game. This significantly adds to the computational complexity of the problem.

Limited learning trials. The simulations are performed on the Minecraft platform, which usually takes several seconds for one episode of game. As a result, it is extremely time consuming to run a large number of simulations. For example, if one episode runs for 5 seconds, it would take around 6 days to train $100,000$ episodes.[1]

Solution Approach

In this section, we present key ideas behind HogRider. First, we design an agent type hypothesis framework to update the belief over the types of agent BLUE, which is robust to uncertainty in agent action observation. On top of that, we propose a novel Q-learning approach to learn an optimal policy against each type of agent BLUE.

Belief Update for Agent Types

Formally, we denote a set of agent types as $\mathcal{T} = \{T\} = \{Random, Focused\}$. We have a prior probability distribution over the agent types, i.e., $P(Random) = 0.25, P(Focused) = 0.75$. At each position, we also know the conditional probability $P(a|T)$ of a type T agent taking movement $a \in \{MF, TL, TR\}$. According to the Bayes theorem, the posterior probability of an agent being type T when it takes movement a is:

$$P(T|a) = \frac{P(a|T) \cdot P(T)}{P(a)}, \tag{1}$$

where $P(a)$ is the probability of movement a being taken.

Generalizing Bayes theorem. As the movements of agent BLUE might be incorrectly observed with a probability of $P_u \approx .25$. We now derive the belief update rule which considers the uncertainty in observing agents' actions.

[1] As we will introduce in the next section, this number is smaller than half of the state-action space size. An alternative to the Minecraft platform is to build our own simulator by formalizing a model for the game. However, the uncertainties discussed above cannot be accurately modelled.

Theorem 1. *With the existence of uncertainty in observing the action of agent BLUE, the agent belief update rule is*

$$\begin{aligned} P(T|a) &= \frac{P(T)\cdot[(1-P_u)\cdot P(a|T)+P_u\cdot P(a|u)]}{(1-P_u)\cdot\sum_{T\in\mathcal{T}} P(a|T)\cdot P(T)+P_u\cdot P(a|u)} \\ &= P(T)\varphi(T,a) \quad \textit{(for simplicity)} \end{aligned} \tag{2}$$

where $P(a|u)$ is the conditional probability of movement a being observed when the observation is incorrect.

Proof. With observation uncertainty, we rewrite $P(a|T)$ as

$$P'(a|T) = (1-P_u)\cdot P(a|T) + P_u \cdot P(a|u). \tag{3}$$

Similarly, the probability $P(a)$ of an action being taken is reformulated as

$$P'(a) = (1-P_u)\cdot \sum_{T\in\mathcal{T}} P(a|T)\cdot P(T) + P_u\cdot P(a|u). \tag{4}$$

Substituting Eqs.(3)-(4) to Eq.(1), we obtain Eq.(2). □

Squash Bayes update with hyperbolic tangent function. Even with the generalization to Bayes theorem, a single wrong action observation could drastically drop the posterior probability of the correct agent type. To mitigate this issue, we propose a more conservative update method, which uses a shifted hyperbolic tangent function $sq(x) = \tanh(x-1)+1 = \frac{e^{2(x-1)}-1}{e^{2(x-1)}+1}+1$ to squash the factor $\varphi(T,a)$ to a value between 0.5 and 2, such that the agent type belief is updated to a new value within $(0.5, 2)$ times of the original value. Formally, the Squash Bayes update function is:

$$P(T|a) = P(T)\varphi'(T,a), \tag{5}$$

where

$$\varphi'(T,a) = \begin{cases} sq(\varphi(T,a)), & \varphi(T,a) \geq 1 \\ \frac{1}{sq(1/\varphi(T,a))}, & \varphi(T,a) < 1 \end{cases}. \tag{6}$$

Adapt Q-Learning to Various Challenges

Q-learning (Watkins 1989) is a model-free reinforcement learning technique for finding optimal policies in sequential decision making problems. We adapt Q-learning to deal with various complexities in the problem. Formally, we formulate the Minecraft game as a Markov Decision Process (MDP). The *state* $\mathbf{s} \in \mathcal{S}$ is defined as a four-dimension vector, which includes the position and facing direction of agent RED, the position and facing direction of agent BLUE, the position of the pig (the pig's behavior does not depend on its facing direction), as well as the steps that have already been taken. The action $A \in \mathcal{A}$ of the MDP is a tuple $A = \langle w, a\rangle$, where w is a binary variable indicating whether HogRider needs to wait for a certain time period (usually 3 seconds), and a is the legal movement after the wait option.[2] A *Q-function* maps a state action pair $(\mathbf{s}, A)$ into real values:

$$Q : \mathcal{S}\times\mathcal{A} \to \mathcal{R} \tag{7}$$

A *policy* $\pi(A|\mathbf{s})$ is a mapping from any state s to an action A that should be taken in that state. In Q-learning, an important challenge is to perform the "exploitation" of the current optimal action, while balancing the "exploration" of potentially better actions. For example, in the ε-greedy policy, it selects the action with the largest Q-value (i.e., $A^* = \arg\max Q(\mathbf{s}, A)$) with probability $1-\varepsilon$, and selects a random action with probability ε.

Usually, Q-learning starts with a randomized initialization of Q-function $Q(\mathbf{s}, A)$. It then generates an *episode* (e.g., one round of game on the Minecraft platform) by performing actions derived from the policy, while updating Q-function with a *stochastic gradient descent* method, using the state-action values returned from the generated episode. Q-learning is guaranteed to converge after sufficient observations of each action at each state (Watkins 1989).

State-Action Abstraction. For each agent, the number of its feasible positions is $25-4 = 21$ and there are 4 directions for each position. For the pig, since it can stay at the two exits, the number of feasible positions is 23. Consequently, the possible combinations of the agents and the pig's positions are $21\times 4\times 21\times 4\times 23 = 162,288$. Taking the last dimension of state (i.e., the number of steps have already been taken) into consideration, the possible number of states is $162,288\times 25 = 4,057,200$, which is a huge number considering the average running time for an episode. While playing the game, we notice that:

It is the relative positions (i.e., distances) between the agents, the pig and the exits, rather than the absolute positions, that determines a player's strategy.

We propose a state abstraction method based on the intuition. An abstracted state has five dimensions, i.e., $PigStat$ which indicates whether the pig is uncatchable ($PigStat = 0$), catchable by a single agent ($PigStat = 1$) or catchable by 2 agents ($PigStat = 2$), $d(R,P)$ which indicates the distance between agent RED and the pig, $d(B,P)$ which denotes the distance between agent BLUE and the pig, $d(R,E)$ which denotes the distance between agent RED and the nearest exit, and $step$ which denotes the number of steps that have been taken. Since $PigStat$ has three possible states, the distance of an agent to the pig ranges from 0 to 9, and the distance of agent RED to the nearest exit ranges from 0 to 7, the total number of states in the abstracted state representation is $3\times 10\times 10\times 8\times 25 = 60,000$, which is less than 15% of the original state space. To transform from the original representation of states to the new one, we only need to calculate the three types of distances based on the original state representation, the calculation overhead of which is trivial.

One issue raised from the abstracted state representation is that, in the new state space, we ignore the direction of agent RED, making it infeasible to directly decide its legal movement. To handle this issue, we propose a corresponding action representation, i.e., a binary variable w which is the same as the original action representation, and a binary variable a' which indicates whether to chase the pig or to head for the exit. To translate the new action representation to the original one, we only need to relate to the position of agent RED, the computation overhead of which is also negligible. Moreover, the action space is reduced from $2\times 3 = 6$ to $2\times 2 = 4$. With new representations of states and actions,

[2] Note that action is different from the legal *movement* in the sense that it has one more dimension of variable w.

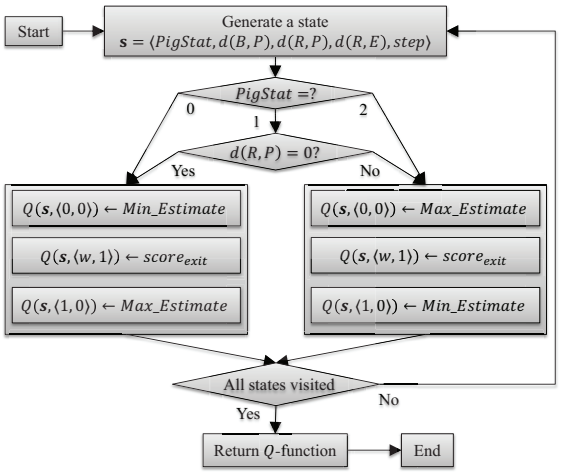


Figure 4: Flow of the human reasoning-derived decision tree

we can reduce the size of the tabular Q-function to less than 1%, with only $240,000$ state-action pairs.

Warm Start Using Human Reasoning. As analyzed above, even after abstraction, there are still $240,000$ state-action pairs. In traditional Q-learning, where Q-function is usually arbitrarily initialized, it is required to run millions of episodes to ensure convergence, which takes days due to the long simulation time. Human demonstration (Wang and Taylor 2017) was proposed to improve efficiency of reinforcement learning, where humans directly assign values to initialize the Q-function. However, human demonstration becomes rather inefficient when there are too many state-action pairs. We take a further step by utilizing a human reasoning-derived decision tree for Q-function's warm start.

The key principles of the warm start approach include 1) when making instant decisions (i.e., w=0), we ignore the pig's movement according to Figure 3(a); 2) we wait for the pig to move when it is at an uncatchable position; and 3) we always choose the action with the highest estimated score. Figure 4 shows the flow of the decision tree for playing with the *Focused* agent. It starts with generating state $\mathbf{s} = \langle PigStat, d(B,P), d(R,P), d(R,E), step \rangle$. We first check the value of $PigStat$. If $PigStat = 1$, we further check distance $d(R,P)$. Then we initialize the Q values of different actions accordingly. For simplicity, we use $w = 0$ (or 1) to represent an action without (or with) waiting time and $a' = 0$ (or 1) to represent chasing the pig (or leaving the game) for $A = \langle w, a' \rangle$. For state $\mathbf{s}$, we can compute the probability $N_m(0)$ (and $N_m(1)$) that pig stays unmoved with $25-step$ steps remaining for $w = 0$ (and $w = 1$). $score_{exit}$ is what agent RED can get by leaving the game through exits. $Min\_Estimate$ is the estimated score when the pig stays unmoved with probability $N_m(0)$ for $PigStat = 0$ (or moves away with probability $1 - N_m(1)$ for $PigStat > 0$); $Max\_Estimate$ is the score for agent RED when the pig is caught at its current position (or the nearest catchable position). We initialize the Q-function for playing with *Random* agent in a similar way.

Active-ε-Greedy. To explore a potentially better action, classical tabular Q-learning methods usually explore the entire action space (e.g., ε-greedy). Counter-intuitively, experimental results (to be explained later in Figure 6(a)) show that the performance of ε-greedy (the line with solid circle markers) declines in the early stage of learning. This is because the optimal actions estimated by our constructed decision tree are usually very good. Therefore, exploring suboptimal actions would lead to performance deterioration in a short term. Meanwhile, if we always deterministically exploit an action that has the highest Q-value, there would be very limited improvement to the current policy. To this end, we propose an *active-ε-greedy* approach to balance the exploration and exploitation. In active-ε-greedy, we still select the optimal action based on the Q-function with the probability of $1 - \varepsilon$. Different from ε-greedy, we explore sub-optimal actions with probability ε, when the Q-value of that action is no less than certain percentage (hand tuned as 50% in practice) of the highest Q-value in the same state.

Training HogRider. In the training process, the type of agent BLUE is fixed as either *Random* or *Focused*. As shown in Algorithm 1, training starts with initialization via the decision tree derived from human reasoning. It then enters the outer loop (Lines 2-8). In each iteration (episode) of the outer loop, HogRider first initializes the state $\mathbf{s}$ (Line 3). It then enters the inner loop (Lines 4-7). In each iteration (step) of the inner loop, HogRider first chooses an action A according to the active-ε-greedy method(Line 5). It then takes the action (after that, agent BLUE will alternatively take an action) and observes the immediate reward R and the next state $\mathbf{s}'$ (Line 6), which will be assigned as the starting state of the next step (Line 7). The inner loop terminates when a terminal state is reached. After that, Q-values for all the state-action pairs $(\mathbf{s}, A)$ that have been visited during this episode are updated (Line 8), where α is the learning rate. $V(\mathbf{s}, A) = \sum_{\mathbf{s}'=\mathbf{s}}^{terminal} R(\mathbf{s}', A)$ denotes the "real" value of the state-action pair $(\mathbf{s}, A)$), which is the sum of immediate reward of all the state action pairs that are visited after state $\mathbf{s}$ during the episode. The entire outer loop terminates after M (e.g., $3,000$) episodes.

Algorithm 1: Training HogRider

1 $Q(\mathbf{s}, A) \leftarrow$ human reasoning, $\forall \mathbf{s}, A$;
2 **while** *#episodes* $\leq M$ **do**
3 Initialize state $\mathbf{s}$;
4 **while** $\mathbf{s}$ *is not terminal* **do**
5 Choose A with active-ε-greedy method;
6 Take action A, observe $R(\mathbf{s}, A)$ and $\mathbf{s}'$;
7 $\mathbf{s} \leftarrow \mathbf{s}'$;
8 $Q(\mathbf{s}, A) \leftarrow (1-\alpha)Q(\mathbf{s}, A) + \alpha V(\mathbf{s}, A)$, $\forall$ visited $(\mathbf{s}, A)$
9 **return** $Q(\mathbf{s}, A)$, $\forall \mathbf{s}, A$;

HogRider

After training, we obtain Q-functions for both agents *Random* and *Focused*, which are denoted as Q_R and Q_F, respectively. We now present HogRider (Algorithm 2) which integrates the learned type-specific Q-function with the agent

type hypothesis framework. Note that unlike the training process, agent BLUE can be either *Random* or *Focused* for different episodes of games. At the beginning of each episode of game, HogRider initializes the belief of agent BLUE being *Random* as $P(Random) = 0.25$ (Line 1). At the same time, HogRider observes the initial state $\mathbf{s}$ of the game and takes the first action according to the initial belief (Line 3). After that, it observes the new state $\mathbf{s}'$ (Line 4) and enters the while loop (Lines 5-9).

Algorithm 2: HogRider

```
1 P(Random) ← 0.25;
2 Observe initial state s;
3 Take action A w.r.t. s and P(Random);
4 Observe s′;
5 while s′ is not terminal do
6 |  Observe action of agent BLUE;
7 |  Update P(Random) using Eq.(5);
8 |  Take action A w.r.t. s′ and P(Random);
9 |  s ← s′ and observe new state s′;
```

Different strategies of selecting an optimal action. In each iteration, it observes the action of agent BLUE in last step (Line 6) according to the difference between $\mathbf{s}$ and $\mathbf{s}'$, updates the belief accordingly (Line 7), takes an action A based on the current state $\mathbf{s}'$ and $P(Random)$ (Line 8). There are three ways to select an action. One way (called `Separate`) is to compare $P(Random)$ with a threshold P_0, and choose $A^* = \arg\max Q_R(\mathbf{s}, A)$ when $P(Random) \geq P_0$ (vice versa). The second way (called `Weighted`) is to maintain an expected Q-value for each action A: $\bar{Q}(\mathbf{s}, A) = P(Random) \cdot Q_R(\mathbf{s}, A) + P(Focused) \cdot Q_F(\mathbf{s}, A)$, and select $A^* = \arg\max \bar{Q}_R(\mathbf{s}, A)$. These two methods are deterministic. The last way (called `Mixed`) is to use $A^* = \arg\max Q_R(\mathbf{s}, A)$ with a probability $P(Random)$, and use $A^* = \arg\max Q_F(\mathbf{s}, A)$ with a probability of $P(Focused)$, which is non-deterministic. After taking an action, it stores the current state in $\mathbf{s}$ and observes a new sate $\mathbf{s}'$ (Line 9). This process until $\mathbf{s}'$ is a terminal state.

Experimental Evaluations

In this section, we present the challenge results of the top 5 teams in MCAC 2017. We also conduct extensive experiments to evaluate performance of various ideas from the proposed solution algorithm.[3]

MCAC Results Table 1 shows the performance of the top 5 teams in MCAC 2017. In terms of both per step mean score and variance/mean, HogRider achieves the best performance. Compared to the second best team, HogRider wins with an edge of 13% in mean score, and an edge of 21.7% in variance/mean. This demonstrates that HogRider outperforms other teams in both optimality and stability.

Table 1: Performance of different teams in MCAC. ↑ means the higher, the better, while ↓ means the opposite. The best performance is highlighted for all the tables.

Team	Mean (↑)	Variance	Variance/Mean (↓)
HogRider	2.752	2.55	0.9266
The Danish Puppeteers	2.435	2.88	1.1828
Bacon Gulch	1.681	2.18	1.2968
Village People	1.618	2.05	1.2670
AASMA	0.591	1.07	1.8105

Strategies for Selecting the Optimal Action As we described, there are three strategies to select an optimal action based on the Q-function, i.e., `Separate`, `Weighted` and `Mixed`. Table 2 shows the performance of the three different strategies, where `Separate` has the best performance in both optimality and stability. In the following experiments, we utilize `Separate` as the base strategy for optimal action selection.

Table 2: Strategies for selecting the optimal action

	Mean score (↑)	Variance	Variance/Mean (↓)
Separate	2.752	2.55	0.9266
Weighted	2.527	2.36	0.9339
Mixed	2.478	2.81	1.1340

Belief Update Table 3 evaluates the performance of HogRider using different methods for updating the beliefs over agent types, where Bayes means traditional Bayes theorem in Eq.(1), G-Bayes means the extension of Bayes theorem in Eq.(2), and G-Bayes+tanh means G-Bayes with a hyperbolic tangent squashing function in Eq.(5).

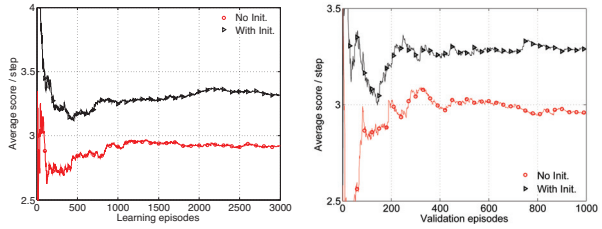
Table 3: Different methods for agent type belief update

Method	Accuracy	Mean(↑)	Var.	Var./Mean(↓)
Bayes	0.578, 0.955 (0.672)	2.456	2.45	0.9976
G-Bayes	0.701, 0.886 (0.747)	2.505	2.27	0.9062
G-Bayes+tanh	0.961, 0.863 (0.937)	2.752	2.55	0.9266

Note that for the Accuracy column, the three numbers are respectively the detection accuracies of two agent BLUE types and the expected overall accuracy. In general, G-Bayes+tanh has the highest overall detection accuracy and optimality among all methods, while being slightly inferior to G-Bayes in stability. Compared with the second best approach, G-Bayes+tanh is better with an edge of 9.9% in optimality, and an edge of 25% in overall detection accuracy.

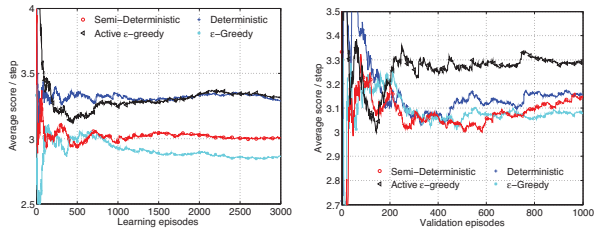
Warm Start Figure 5 shows the learning curve and validation curve of HogRider with human reasoning-aided initialization playing with the *Focused* agent. The validation curve shows that, using initialization, the mean score of HogRider

[3]There are two ways to measure the performance of different methods, i.e., per step score and per game (episode) score. Due to page limit, we only show results for per step scores, while we refer to our online appendix (Xiong et al. 2017) for results of per episode scores, as well as figures showing the detailed curves of the results that cannot be presented with tables.



(a) Learning curve (b) Validation curve

Figure 5: Evaluate human reasoning-aided initialization



(a) Learning curve (b) Validation curve

Figure 6: Evaluate different exploration methods

significantly improves by 10.9%, while variance/mean decreases by 5.7%. The learning curve indicates that we can learn a better Q-function much faster.

Different Exploration Methods Figure 6 compares the learning curve and validation curve of the active-ε-greedy with other exploration methods (playing with $Focused$ agent), where Deterministic always selects the current optimal action $A^* = \arg\max Q(\mathbf{s}, A)$), ε-greedy selects A^* with probability $1 - \varepsilon$ and randomly explores other actions with probability ε, Semi-Deterministic selects A^* with probability $1 - \varepsilon$ and explores the second best action with probability ε. From the learning curve, we can see that 1) arbitrary exploration (i.e., ε-greedy) can be rather harmful when learning trials are limited, 2) deterministic methods provide limited help in improving the current policy, 3) semi-deterministic methods always explore the second-best action without considering the estimated value of it, which is also inefficient, and 4) with active-ε-greedy, the potential optimal actions (with at least 50% estimated value of the best action) are exploited and the policy keeps improving along the training process. The validation curve demonstrates the effectiveness of active-ε-greedy method. In terms of mean score, active-ε-greedy is better with an edge of 4.1% compared with the second best approach (Deterministic), and better with an edge of 6.7% compared with ε-greedy.

Compare HogRider with Human Players We also invited 10 human players from our university to play the Pig Chase game, all of whom are PhD candidates. Surprisingly, the results show that HogRider performs even better than human players with an edge od 28.1% in mean increase and 29.6% in variance/mean decrease.

Table 4: Comparison with human players

Approach	Mean(↑)	Variance	Variance/Mean(↓)
HogRider	2.752	2.55	0.9266
Human	2.149	2.83	1.3169

Lessons Learned

Developing HogRider has been a progressive experience. Different from the constructive flow that presents HogRider, it was rather a step by step process where new discoveries and ideas interchangeably emerge. We gained several insights into solving challenging problems in the areas of collaborative AI, as well as other more general research domains. **First, "if you know yourself and your enemy, you will never lose a battle".**[4] It has been a long exploration before we set up the integrated framework of agent type hypothesis and Q-learning. Our initial plan was to simply learn an effective policy with Q-learning without differentiating different types of agents. Despite the model-free property of Q-learning, its direct application worked poorly simply because we did not exploit the problem structure. While looking for cutting-edge tools is important, learning the foundations of the problem ensures the right direction. **Second, human intuition can take machines to the heights.** Due to its popularity, we initially decided to use DQN (Glorot and Bengio 2010) as the learning approach to learn policies against each agent type. Despite its power in expressing Q-functions, initialization does not work due to the parameterized Q-functions, which poses DQN at a disadvantageous place against tabular Q-learning approaches. In fact, a few teams using DQN had poor performance in the challenge. It is rather surprising and enlightening how much human reasoning could help. This type of initialization could be utilized in other areas where domain knowledge can help derive human reasoning. **Moreover, models and solution algorithms should be kept updating with new discoveries of latent properties.** For example, the uncertainty in agent action observation was not discovered until the algorithm was almost built. To this end, we made two additional adaptations to the traditional Bayes theorem which brought significant improvement to the algorithm.

Conclusion & Future Research

There are many characteristics such as complex interactions among agents, uncertainties, and limited learning trials that make effective sequential planning extremely challenging. The MCAC, which is designed to encourage research relating to various problems in Collaborative AI, is a typical test bed with all these challenges. We introduce HogRider, champion agent of MCAC in 2017, which has two key innovations. First, we design a generalized agent type hypothesis framework to handle the uncertainties in agent type and observation in the other agent's action. Second, we propose a novel Q-learning approach to learn effective policies against each type of collaborating agents with various novel ideas, including state-action abstraction, a human reasoning-aided

[4] It is from an ancient Chinese military treatise, *The Art of War*.

approach to initialize Q-function, and an active greedy strategy to balance the exploitation and exploration.

One potential future research is to look at scenarios where agents are completely unknown to each other before they meet. For example, there are no prior knowledge of the distribution of agent types, or there are no pre-defined types for the collaborating agents. Another potential research is to propose solution algorithms which can be generalized to different types of environments. For example, if the positions of the blocks are replaced, more blocks are added, or the size of grids are varied, how could the policy learned from the old setting applied to the new setting? In these scenarios, algorithms might have to combine off-linely trained policies with online learning (Anderson 2008), which resembles the idea of Endgame solving (Ganzfried and Sandholm 2015) in Computer Poker games, and the set of transfer learning algorithms that are designed for reinforcement learning domains (Taylor and Stone 2009).

Acknowledgement

We thank Microsoft Research for organising the challenge. This research is supported by the National Research Foundation, Prime Ministers Office, Singapore under its IDM Futures Funding Initiative, and is supported by NRF2015 NCR-NCR003-004 and NCR2016NCR-NCR001-002.

References

Agmon, N., and Stone, P. 2012. Leading ad hoc agents in joint action settings with multiple teammates. In *AAMAS*, 341–348.

Albrecht, S. V.; Crandall, J. W.; and Ramamoorthy, S. 2016. Belief and truth in hypothesised behaviours. *Artificial Intelligence* 235:63–94.

Anderson, T. 2008. *The Theory and Practice of Online Learning*. Athabasca University Press.

Barrett, S.; Stone, P.; Kraus, S.; and Rosenfeld, A. 2013. Teamwork with limited knowledge of teammates. In *AAAI*, 102–108.

Barrett, S.; Stone, P.; and Kraus, S. 2011. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *AAMAS*, 567–574.

Brafman, R. I.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2009. Planning games. In *IJCAI*, 73–78.

Conitzer, V., and Sandholm, T. 2007. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning* 67(1-2):23–43.

De Cote, E. M.; Chapman, A. C.; Sykulski, A. M.; and Jennings, N. R. 2012. Automated planning in repeated adversarial games. *arXiv preprint arXiv:1203.3498*.

Dunne, P. E.; Kraus, S.; Manisterski, E.; and Wooldridge, M. 2010. Solving coalitional resource games. *Artificial Intelligence* 174(1):20–50.

Foerster, J.; Assael, Y. M.; de Freitas, N.; and Whiteson, S. 2016. Learning to communicate with deep multi-agent reinforcement learning. In *NIPS*, 2137–2145.

Ganzfried, S., and Sandholm, T. 2015. Endgame solving in large imperfect-information games. In *AAMAS*, 37–45.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 249–256.

Grosz, B. J., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86(2):269–357.

Horák, K.; Bosanskỳ, B.; and Pechoucek, M. 2017. Heuristic search value iteration for one-sided partially observable stochastic games. In *AAAI*, 558–564.

Horling, B.; Lesser, V.; Vincent, R.; Wagner, T.; Raja, A.; Zhang, S.; Decker, K.; and Garvey, A. 1999. The TAEMS white paper.

Jordán, J., and Onaindia, E. 2015. Game-theoretic approach for non-cooperative planning. In *AAAI*, 1357–1363.

Jordán, J.; de Weerdt, M.; Onaindia, E.; et al. 2017. A better-response strategy for self-interested planning agents. *Applied Intelligence* 1–21.

Kuno, N., and Schwiderski-Grosche, S. 2017. Presenting the winners of the Project Malmo Collaborative AI Challenge. https://www.microsoft.com/en-us/research/blog/malmo-collaborative-ai-challenge-winners/.

Mojang. 2017. Minecraft. https://minecraft.net/en-us/.

Sandholm, T. 2015. Abstraction for solving large incomplete-information games. In *AAAI*, 4127–4131.

Seuken, S., and Zilberstein, S. 2008. Formal models and algorithms for decentralized decision making under uncertainty. In *AAAMAS*, 190–250.

Singh, S.; Kearns, M.; and Mansour, Y. 2000. Nash convergence of gradient dynamics in general-sum games. In *UAI*, 541–548.

Southey, F.; Bowling, M. P.; Larson, B.; Piccione, C.; Burch, N.; Billings, D.; and Rayner, C. 2012. Bayes' bluff: Opponent modelling in poker. *arXiv preprint arXiv:1207.1411*.

Stone, P.; Kaminka, G. A.; Kraus, S.; Rosenschein, J. S.; et al. 2010. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI*, 1504–1509.

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7:83–124.

Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10:1633–1685.

Wang, Z., and Taylor, M. E. 2017. Improving reinforcement learning with confidence-based demonstrations. In *IJCAI*, 3027–3033.

Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, King's College, Cambridge.

Xiong, Y.; Chen, H.; Zhao, M.; and An, B. 2017. Hogrider appendix. http://AAAImalmo.weebly.com.