Learning by Reusing Previous Advice in Teacher-Student Paradigm

Changxi Zhu
School of Software Engineering
South China University of Technology
cxzhu.cn@gmail.com

Yi Cai
School of Software Engineering
South China University of Technology
ycai@scut.edu.cn

Ho-fung Leung
Department of Computer Science and Engineering
The Chinese University of Hong Kong
lhf@cuhk.edu.hk

Shuyue Hu
Department of Computer Science
National University of Singapore
dcshus@nus.edu.sg

ABSTRACT

Reinforcement Learning (RL) has been widely used to solve sequential decision-making problems. However, RL algorithms suffer from poor sample efficiency and require a long time to learn a suitable policy, especially when multiple agents are learning without prior knowledge. This problem can be alleviated through reusing knowledge from other agents during the learning process. One notable approach is advising actions based on a teacher-student relationship, where the decision of a student agent during learning is aided by an experienced teacher agent. A critical assumption in teacher-student paradigm is that the communication may be limited, so that a student may wait for a while and learn by itself before receiving the next advice. More importantly, in some noisy or stochastic environments, the student may not be able to master the advised actions when they are only performed once. We propose three methods for agents choosing between learning by exploration, asking for advice and reusing previous advice. The results show that our approaches significantly outperform existing advising methods without reusing advice.

KEYWORDS

reinforcement learning; multi-agent; action advising; teacher-student

ACM Reference Format:
Changxi Zhu, Yi Cai, Ho-fung Leung, and Shuyue Hu. 2020. Learning by Reusing Previous Advice in Teacher-Student Paradigm. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020*, IFAAMAS, 9 pages.

1 INTRODUCTION

Reinforcement learning (RL) has been employed to solve many real world problems, e.g., robotics, optimizing memory control and personalized web services [5, 15]. However, RL often suffers from slow learning speed in complex applications. This can be further intensified when multiple agents are independently learning and observing, since each of them needs to adapt for others. Undoubtedly, when there exists a trained agent or a human expert, a new agent can benefit from asking about the solution for current task. A more general situation is that all agents learn without prior knowledge. Nevertheless, they can accelerate the learning process by sharing currently acquired knowledge with each other. Another practical consideration is that the computing resource or communication may be limited. It is essential for agents to decide what and when to share, as well as how to utilize the shared knowledge.

Recently, the teacher-student framework [26] has received lots of attention. In this paradigm, a trained agent (named teacher) advises a learning agent (named student) on which action to take in a state. The teacher and the student are only required to have the same action set without specifying their learning structures, making this framework quite flexible for realistic scenarios, for example, agents equipped with different sensors or policy representations collaborate together. Da Silva et al. [8] adapt the teacher-student framework to a Multi-Agent System (MAS) composed of several simultaneously learning agents, focusing on how action advising affects agents' mutual learning processes. In their work, the advising opportunities are established on demand, which considers the communication cost among agents. Omidshafiei et al. [17] view teaching in Multi-Agent Reinforcement Learning (MARL) as a high-level sequential decision task. An agent who takes the role of student learns to ask for advice or not. Another agent who takes the role of teacher learns to advise all possible actions in states, which supports teaching heterogeneous teammates.

Previous works on action advising mainly address the problems of when and what to advise, while rarely discuss the problem of how to use the advice more effectively. A common key assumption [2, 8, 26] behind the teacher-student framework and its variations is that an agent who takes the role of teacher is more experienced than another agent who takes the role of student in certain states. The teacher is assumed to be an expert or has explored these states plenty of times. However, the advice shared from teacher is performed only once, and then will be forgot. Imagine that a coach teaches a rookie how to shoot in a soccer game. The rookie will be encouraged if he aims the shoot by following his coach's advice in current position. He may also miss the shot at this time due to noisy or stochastic environment. In such case, if the rookie ignores the former suggestion, he is likely to try other options in that position or wait for a moment until obtaining next advice from the coach. Then the rookie needs to take more time to perform as competitive as the coach. However, we consider that if previous advice is reused,

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

the rookie's learning can be accelerated since it will spend less time to explore current situation.

Our work focuses on how a student (e.g., the rookie) can benefit from reusing previous advice, in order to effectively utilize the shared knowledge from a more experienced teacher (e.g., the coach). As long as a student receives an advice in a state, the state will be labeled as "advised" and the advice pair $\langle state, advice \rangle$ will be stored. When the student visits such advised state again, it may remember the advice previously shared from a teacher. Considering that agents are initiated without prior knowledge, their policies are generally non-optimal during learning. Insisting on some non-optimal advices for too long may hinder the overall performance. A student should choose proper time to reuse advised actions. Occasional exploration for the students also needs to be allowed so as to surpass the teachers. In this paper, we propose three methods to help an agent choose between learning by exploration, asking for advice, and reusing the shared knowledge at every time step. In *Q-change per Step* (QChange), an agent will reuse previous advice if such advice leads to an increased Q-value. Since the environment can be noisy or stochastic, the reusing procedure in QChange may not have a long term impact on the advised states. *Reusing Budget* (ReBudget) is proposed for each agent reusing the advised actions in a fixed number of times. As the advice from teacher may be outdated during learning, we also propose *Decay Reusing Probability* (Decay) to allow agents learning in usual advising frameworks while reusing previous advices with decaying probabilities.

Our contribution in this paper is threefold. First, to our best knowledge, this is the first paper to address the problem of how to reuse the advice from teachers. Second, we propose three methods that allow agents learning by reusing previous advice upon usual action advising frameworks under limited communication. Third, we show that reusing advice significantly accelerate learning in one single learning agent game Mario, and two multiple learning agent problems: Predator-Prey and Half Field Offense.

2 RELATED WORK

The idea of peer to peer knowledge sharing has its root in RL. Tan [24] points out that Q-learning agents can communicate several aspects of their learning processes, such as policies (or value functions), episodes (state, action quality triplets) and sensations, in order to accelerate the joint learning. *Q-values Sharing* has shown to surpass advising actions both in single-agent [11] and multi-agent tasks [29] under limited communication. However, it requires that agents have identical optimal polices and learning structures or to build hand-coded knowledge mapping relationship, resulting in limit scalability in complex applications. Works on *imitation learning* [3, 19] study how a student learns a policy by observing an experienced agent or a human's actuation. *Learning from Demonstration* [4] tries to mimic the collected demonstrations, which are typically recorded as state-action pairs following an expert's policy for a certain period of time. By contrast, our work builds upon *Action Advising* [2, 8, 10, 26], where an agent should act to maximize a reward with the help of interweaved action suggestions, not just to mimic a teacher. Advising actions is one of the most flexible knowledge sharing frameworks, as agents need only a common understanding of the actions and current state. The main idea involves two fundamental roles, a teacher and a student. An experienced agent or a human expert who takes the role of teacher guides the action selection of a learning agent who takes the role of student by suggesting actions to take in certain states. Existing literature on action advising mainly emphasizes to solve the problems of when and which action to advise, while our work focuses on how to use the advice more efficiently. As communication among agents is usually limited or costly, it is critical to decide advising opportunities. There have been three modes of giving advice: a) student-initiated [7]; b) teacher-initiated [26]; c) jointly-initiated [2]. Advising decisions made only by a student may be weak as itself is still learning. By contrast, the teacher-initiated method requires that the teacher constantly monitor the student's learning progress. The jointly-initiated method establishes initiating advising relations under the agreement of both teacher and student, which is quite suitable for applications with limited communication.

The solution to the problem of which action to advise can be classified as two lines. In the first one [2, 8, 9, 26, 30], when a student asks for advice, a teacher shares its currently learned best action (or optimal action), which corresponds to the maximum Q-value for the requested state. The intuition behind this method is that the student and the teacher have the same (or similar) learning goal (e.g., goal states). However, there exist scenarios where agents learn best by exhibiting behavioral specialization or where agents have heterogeneous capabilities and state-action space altogether. They may learn different optimal polices, so that advising personally best actions even degrades team performance. Omidshafiei et al. [17] introduce LeCTR that enables a student to learn whether asks for advice or not, and a teacher to learn which action to advise. The current task is transformed into two level RL problems: the task-level and the teaching-level. In the teaching-level, agents are equipped with a centralized actor-critic algorithm to learn both when and what to advise. They also explore how to adapt LeCTR in domains with long horizons, delayed rewards, and continuous states/actions [13]. However, compared with heuristic methods, both works demand extensive interactions with the task-level learning process, in order to evaluate the teacher's advice.

Another category of action advising on existing works is based on the task they apply in: a) discrete states/actions [8, 17, 25]; b) continuous states and discrete actions [12, 26]; c) continuous states/actions [13]. The environment with discrete states/actions, often based on grid game, is consider to be the most commonly applied in RL tasks. Da Silva et al. [8] propose several heuristics extended from [26] to decide when to ask for and give advice in a cooperative team of SARSA(λ) agents. İlhan et al. [12] focus on solving how to evaluate agents' confidence in the environment with continuous states and discrete actions, where multiple DQN agents learn together. In the work of Kim et al. [13], agents use deep models in their task-level policies and act in environment with continuous states/actions, while only two agents are involved.

3 PRELIMINARIES

3.1 Single-agent RL and MARL

RL is used to solve decision-making problems, which are usually formulated as Markov Decision Processes (MDPs) [23]. An MDP is described by a tuple $\langle S, A, T, R, \gamma \rangle$, where S is a set of environment

states, A is the available actions of an agent, T is transition function, R is reward function, and γ is discount factor. At each time step, an agent observes current state s and selects action a. Then the next state is defined by T and reward signal r can be received. The goal of an agent is to learn a policy $\pi : S \rightarrow A$ which maps states to actions in such a way that the expected cumulative discounted reward is maximized. Temporal difference (TD) RL algorithms such as Q-learning [28] and SARSA [23] learn an action-value function, $Q(s, a)$, which is an estimate of the expected return that an agent takes action a in state s. The Q-function of an agent is incrementally updated based on following rules,

$$Q(s, a) \leftarrow Q(s, a) + \alpha \times \delta \tag{1}$$

where $\alpha \in [0, 1]$ is the learning rate, and δ is TD error. In Q-learning, $\delta = r + \gamma \max_a Q(s', a) - Q(s, a)$, where $\gamma \in [0, 1)$ is the discount factor. In SARSA, $\delta = r + \gamma Q(s', a') - Q(s, a)$, where $a' \in A$ is the next action that the agent will execute. The Q(λ) [23] and SARSA(λ) [20] are extensions of Q-learning and SARSA respectively, which improve the learning speed by updating Q-values from past states with the TD error of current time step. $\lambda \in [0, 1]$ is trace decay factor that controls the impact of current TD error on past Q-values in each episode. As the updating times of previous Q-values increases, current TD error has smaller influence on these Q-values. These algorithms are guaranteed to converge to the optimal Q-function Q^*, from which the optimal policy π^* can be derived,

$$\pi^*(s) = \arg\max_a Q^*(s, a) \tag{2}$$

During learning, the agent executes its best actions corresponding to the maximum Q-values in states. The agent also tries to find even better options. A common strategy is *ϵ-greedy*, where the agent takes a random action with a small probability ϵ and its best action with probability $(1 - \epsilon)$.

In the multi-agent case, we are interested in cooperative RL agents getting local observations and learning in a decentralised fashion [16]. They jointly affect the environment and receive the same reward, while learn individual policies without accessing knowledge beyond the environment.

3.2 Teacher-Student Framework

Our work builds upon usual teacher-student frameworks. Here we introduce a promising multi-agent advising framework AdhocTD [8], where when to share teachers' best actions is well defined. This framework has shown to surpass two branches of method under budget constraint: Episode Sharing [24], agents share tuples of $\langle s, a, r, s' \rangle$ after a successful episode and then these tuples will be used to update corresponding Q-values, and Importance-Based Advising [26], where advice is given if all other agents detect the range of Q-values in a state is higher than a predefined threshold t: $I(s) = \max_a Q(s, a) - \min_a Q(s, a) > t$. In AdhocTD, for current state s, agent a_i asks for advice with an asking probability calculated by function $P_{ask} = (1+v_a)^{-\sqrt{n_{visit}(s)}}$, where v_a is a predetermined parameter, $n_{visit}(s)$ is the number of times that the agent visits state s. For the state of agent a_i, another agent a_j gives its best action with a giving probability calculated by function $P_{give} = 1-(1+v_b)^{-\sqrt{n_{visit}(s)} \times I(s)}$, where v_b is a predetermined parameter. In case agent a_i receives more than one advice, it selects the executed action through a majority vote. In order to model communication cost, the number of times that an agent asks for and gives advice are constrained by two numeric budget b_{ask} and b_{give} respectively.

Algorithm 1 Q-change per Step for agent i

Require: function Φ, asking probability function P_{ask}

```
for each time step t do
    let current state be s_t
    if b_ask > 0 then
        if s_t ∈ Φ then
            a_t ← Φ(s_t)
        if no advice is reused then
            if P_ask(s) > getRandomValue(0, 1) then
                denote a_t as the advice chose by agent i
                b_ask ← b_ask − 1
                Φ(s_t) ← a_t
        if no action is reused or advised then
            select action a_t from usual exploration strategy
        β ← Q_t(s_t, a_t)
        perform action a_t in state s_t
        if s_t ∈ Φ then
            β ← Q_{t+1}(s_t, a_t) − β
            if β <= η then
                remove state s_t from Φ
```

4 METHOD

We focus on a teacher-student relationship, which is established only when needed. The goal of our proposed methods is to explore how a student effectively utilizes the advice previously shared from a teacher, thereby accelerating learning process. Before introducing the method, we assume that agent i has taken the role of student and asked for advice in current state s (e.g., with the probability $P_{ask}(s)$ defined by AdhocTD). Correspondingly, a more experienced agent j has taken the role of teacher and given advice for the state requested by agent i (e.g., with the probability $P_{give}(s)$ defined by AdhocTD). After receiving an advice, agent i labels s as an advised state. When agent i visits state s next time, such label makes it recall that this state has been advised. If agent j has already learned an optimal policy for current task, there is no doubt that the action previously advised by agent j benefits agent i's learning. Despite this, occasional exploration needs to be allowed for agent i to surpass the teacher's performance. In addition, where all agents learn together, their policies are general to be non-optimal. Therefore, agent i should decide whether learning by itself, asking for advice again or reusing previous advice when visiting the advised states. Each agent is limited by a budget b_{ask} to ask for advice and a budget b_{give} to give advice, which is a regular setting in works on action advising [2, 8, 12, 26]. Note that agent i begins to learn by itself immediately when asking budget b_{ask} is used up.

4.1 Q-change per Step

As agent i is still learning, it is not easy to decide when to reuse the advised actions. The agent can evaluate every piece of impact of these actions, and then make a decision. A good advice may be associated with a positive feedback. If agent i is a Q-learner, it can

Algorithm 2 Reusing Budget for agent i

Require: function Φ, remaining budget function Ψ, maximum budget L, asking probability function P_{ask}

for each time step t **do**
  let current state be s_t
  if $b_{ask} > 0$ **then**
    if $s_t \in \Phi$ **then**
      if $\Psi(s_t) > 0$ **then**
        $a_t \leftarrow \Phi(s_t)$
        $\Psi(s_t) \leftarrow \Psi(s_t) - 1$
    if no advice is reused **then**
      if $P_{ask}(s) > getRandomValue(0, 1)$ **then**
        denote a_t as the advice chose by agent i
        $b_{ask} \leftarrow b_{ask} - 1$
        $\Phi(s_t) \leftarrow a_t$
        $\Psi(s_t) \leftarrow L$
    if no action is reused or advised **then**
      select action a_t from usual exploration strategy
    perform action a_t in state s_t

Algorithm 3 Decay Reusing Probability for agent i

Require: function Φ, reusing probability function P_{reuse}, decay value ρ

for each time step t **do**
  let current state be s_t
  if $b_{ask} > 0$ **then**
    $a_{pre} \leftarrow \Phi(s_t)$
    if $P_{reuse}(s_t) > getRandomValue(0, 1)$ **then**
      $a_t \leftarrow a_{pre}$
      $P_{reuse}(s_t) \leftarrow P_{reuse}(s_t) \times \rho$
    if no advice is reused **then**
      if $P_{ask}(s) > getRandomValue(0, 1)$ **then**
        denote action a_t as the advice chose by agent i
        $b_{ask} \leftarrow b_{ask} - 1$
        $\Phi(s_t) \leftarrow a_t$
        $P_{reuse}(s_t) \leftarrow 1$
    if no action is reused or advised **then**
      select action a_t from usual exploration strategy
    perform action a_t in state s_t

reuse the advice next time encountering the same state when the advised action leads to an increased Q-value. The change of the Q-value will be detected after agent i updating its Q-function. We call this method as *Q-change per Step* (QChange), as described in Algorithm 1. After receiving action a suggested by agent j, agent i performs the action in current state s. If the change of the Q-value corresponding to action a in state s exceeds a predefined threshold η, agent i labels state s as an advised state and stores such advice pair $\langle s, a\rangle$ to a function $\Phi : S \rightarrow A$. Due to the budget constraint, the maximum number of stored states is limited to b_{give}. When agent i visits the labeled state again, action a will be queried from Φ and then executed. After that, the policy is updated as usual. In the meanwhile, the assessing procedure of reusing action a will be performed again, which increases the chance of reusing previous advice. If no advice is reused, agent i selects current action according to an usual teacher-student framework (e.g., AdhocTD). In particular, when agent i receives a new advice in state s, the original advised action will be replaced by the latest one in terms of the teachers' latest knowledge.

4.2 Reusing Budget

In QChange, teachers' advice may not have a long term impact on advised states, since the estimation in changes of Q-values can be misled due to noise or stochastic environment. When a student receives an optimal action from a trained agent, insisting on advice for a while is likely to stabilize performance. In *Reusing Budget* (ReBudget), agent i will memorize advice for a fixed number of times. Each advised action will not be reused when a maximum reusing budget L is reached. Algorithm 2 describes how agent i reuses previous advice with ReBudget. After establishing a teacher-student relationship with agent j, agent i performs advice a and labels current state s as advised. The advice pair $\langle s, a\rangle$ is stored to function Φ, and the remaining budget function $\Psi : S \rightarrow R$ for state s is initialized with budget L. Every time agent i encounters state s, action a will be executed instead of selecting action by itself and asking for advice. Then the associated remaining budget for state s in function Ψ will be decremented by 1. During this period, if agent i receives a new advice from other agents in state s, the latest advice will be used and the reusing budget will be reset to L. The reusing procedure in state s is stopped when the remaining budget is equal to 0.

4.3 Decay Reusing Probability

The agent who takes the role of teacher may still be adjusting its policy even if it is considered to be a more experienced agent. As training times increases, previously advised actions are likely to be outdated, especially when all agents are learning simultaneously. Every time agent i visits advised states, a more flexible method is to give it the opportunity to choose between reusing previous advice, learning by itself and asking for advice. We name this method as *Decay Reusing Probability* (Decay). The detail of Decay is shown in Algorithm 3. If agents use ϵ-greedy as exploration strategy, when agent i visits an advised state s, current action a_t at time step t is selected as follows:

$$a_t = \begin{cases} \text{Previous Advice} & \text{w.p. } P_{reuse} \\ \text{Asking for Advice} & \text{w.p. } P_{reuse} \times P_{advice} \\ \text{Greedy Action} & \text{w.p. } (1 - P_{reuse} \times P_{advice}) \times (1 - \epsilon) \\ \text{Random Action} & \text{w.p. } (1 - P_{reuse} \times P_{advice}) \times \epsilon \end{cases}$$

where P_{reuse} is a reusing probability for state s, and P_{advice} is the probability of receiving advice from other agents. In AdhocTD, P_{advice} can be defined as: $P_{advice} = P_{ask} \times P_{give}$. In our work, P_{reuse} determines whether learning agent i should reuse a teacher's advice to guide its action selection: agent i will follow previous advice with probability P_{reuse}, ask for a new advice with probability $P_{reuse} \times P_{advice}$, exploit its Q-values with probability $(1-P_{reuse}\times P_{advice})\times(1-\epsilon)$, and act randomly with probability $(1-P_{reuse}\times P_{advice})\times\epsilon$. When agent i receives an advice in state s, $P_{reuse}(s)$ is typically initialized with 1 and such advice will be stored to function Φ. Note that the value of $P_{reuse}(s)$ will be reinitialized when the next

advice is shared from other agents in state s, ensuring that agent i can follow the teachers' current learning. As agent i repeatedly performs the latest advice in state s, $P_{reuse}(s)$ decays exponentially. We define $P_{reuse}(s)$ for state s as follows:

$$P_{reuse}(s) = \rho^{m_{visit}} \tag{3}$$

where decay value $\rho \in [0, 1]$, and m_{visit} is the number of times that agent i adopts the latest advice in state s and will be refreshed when agent i receives the next advice. When ρ is 0, agent i is not able to reuse advice, which is equal to learn in usual teacher-student framework. When ρ is 1, agent i simply follows a teacher's advice rather than learning a policy. The reusing process will be stopped when the maximum asking budget is reached.

5 EXPERIMENTS

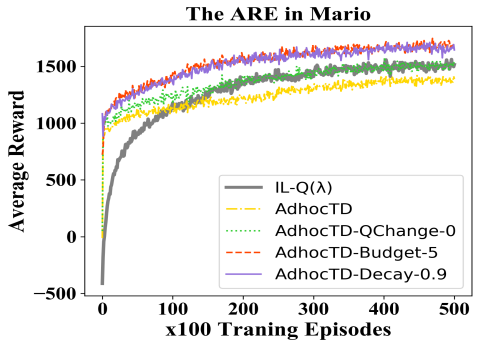
We evaluate our method on both single learning agent case and multiple learning agents case. The former one requires that an experienced agent who has already learned a fixed policy assists the learning of a new agent. By contrast, in the latter one, all agents are learning from scratch and accelerate the joint learning by advising each other. Three RL problems are offered to perform experiments: Mario, Predator-Prey and Half Field Offense. Mario is a complex stochastic game and used as the single agent environment. Predator-Prey is a popular benchmark for multi-agent learning. Half Field Offense is a hard learning robot soccer game, where agents must take stochastic action effects and noisy observations into account. We compare the performance of following methods:

(a) **IL/Multi-IL**. Each agent learns independently without communication, which is served as a baseline to validate the benefit of advising and reusing previous advice in different tasks.
(b) **AdhocTD**. The detailed AdhocTD is described in Section 3.2. This method has shown to largely reduce advising budgets while achieving excellent results in Half Field Offense [8]. We adapt AdhocTD in the single-agent case by recording the number of times that a teacher visits each state and the Q-values of the teacher after converging to a fixed policy.
(c) **AdhocTD-QChange/ReBudget/Decay**. AdhocTD can be viewed as an ideally basic learning algorithm and is combined with our proposed methods QChange, ReBudget and Decay.

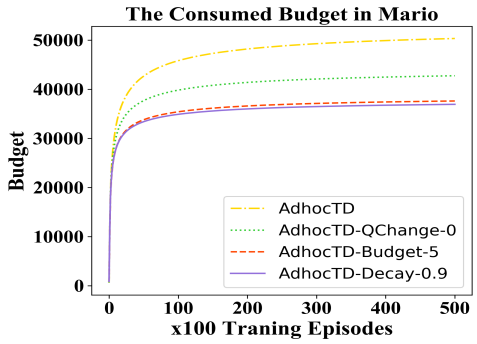
5.1 Mario

Game Description. Mario is a benchmark domain based on Nintendo's Mario Brothers [18]. Our implementation is based on [27], where only one agent learns from scratch, and can take the role of student.[1] The other agent who has trained to converged to a fixed policy is employed to be as a teacher. There are many possible representations of the state-space in this game. In order to test our approaches in a large state-space, 27 discrete state-variables are used, which is also adopted by previous works [22, 27]. The variables encode a learning agent's state/position information, surrounding blocks, and enemies. The agent learns to choose between 12 ($3 \times 2 \times 2$) actions (movement direction $\times$ jump button $\times$ Run/Fire button). Mario is designed as an episodic task. Every episode, an agent plays a randomly generated level, starting from a randomly selected mode (small, large, fire-mario). The level is end either with the agent's success, the agent's death, or a timeout of 200 seconds. For the learning task, both the student and teacher are equipped with Q(λ) with following parameters: α=0.001, γ=0.9, λ=0.5. We use ϵgreedy as action selection strategy for all methods, where ϵ= 0.05.

[1]The source code is available at https://github.com/chauncyzhu/Mario.



(a) ARE of all methods



(b) Budget consuming of all methods

Figure 1: ARE and budget consumption of IL-Q(λ), AdhocTD, AdhocTD-QChange/ReBudget/Decay when b_{ask} = b_{give}=50,000, η=0, L=5 and ρ=0.9

Evaluation Metrics. In Mario, the learning agent's performance is assessed by *Average Reward per Episode* (ARE). ARE is an agent's average reward for each episode during a predefined number of training episodes, and we set 100. The teacher has trained for 50,000 episodes to obtain a fixed policy. Then in all methods (except IL-Q(λ)), the students are advised by the same teacher. We perform 30 runs for all methods to stabilize the performance. The selection of v_a and v_b may heavily effect the advising opportunities in AdhocTD as well as corresponding action reusing methods. When v_a is small and v_b is high, the student is more likely to ask for advice and the teacher is more likely to give advice, resulting higher communication cost. One the contrary, high value of v_a and low value of v_b make the student asks for advice when it visits current state few times and the teacher gives advice when it has experienced the student's state many times respectively. Then we choose v_a=2 and v_b=0.2, in which AdhocTD spends about 50,000 budget in the 50,000 training episodes. In order to compare the effect of different value of parameters on the learning process, we set: a) η=0, 0.01 and 0.03; b) L=5, 10 and 100; c) ρ=0.8, 0.9 and 0.99.

Table 1: Performance metrics for the agents in Mario (average of 30 trials). Maximum values of action reusing methods with η, L and ρ are shown in bold respectively. Higher values are better. The best value of each column is underlined.

Agents	Jump Start	Last	Mean
IL-Q(λ)	-412.686	1516.355	1286.195
AdhocTD	-11.933	1388.411	1238.722
QChange (η=0)	49.667	**1518.773**	**1362.383**
QChange (η=0.01)	36.467	1452.036	1290.747
QChange (η=0.03)	**145.267**	1408.7	1260.696
ReBudget (L=5)	721.677	**1661.853**	**1537.968**
ReBudget (L=10)	1262.8	1634.525	1510.712
ReBudget (L=100)	**1337.125**	1584.468	1426.272
Decay (ρ=0.8)	919.8	**1653.793**	1511.439
Decay (ρ=0.9)	1082.774	1643.75	**1520.172**
Decay (ρ=0.99)	**1722.067**	1540.297	1417.719

Experimental Results. The averaged ARE and the consumption of advising budget over 30 runs for all methods are shown in Figure 1. In Figure 1a, we firstly present the ARE value of action reusing methods turned with best parameters. We can see that action advising schema, AdhocTD has significant higher ARE than IL-Q(λ) before about 12,000 episodes. During this period, the student visits most states very few times, and benefit more from asking for advice. Hence, the student quickly spends advising budgets, as shown in Figure 1b. As the agent learns, it is less likely to ask for advice. AdhocTD gradually has lower ARE than IL-Q(λ), which probably means that the student does not master the advised actions due to stochastic environment. Too much advice even disturbs the student's learning. As for AdhocTD-QChange, since the student is less likely to insist on the teacher's advice for too long, this method finally achieves similar ARE values as IL-Q(λ). By contrast, AdhocTD-ReBudget/Decay have much higher ARE and spend much lower budget than other methods. Furthermore, the consumed budget of AdhocTD-Decay is slightly lower than AdhocTD-ReBudget.

Average Jump Start (the reward of the first training episode), Last (the last value of ARE), and Mean (the average ARE of the whole training episodes) are detailed in Table 1. We can see that AdhocTD-ReBudget with L=5 achieves the highest Mean performance. The advice reusing methods (except QChange with η =0.01 and 0.03) have significantly better Last and Mean performance than AdhocTD and IL-Q(λ). As for the Jump Start, since agents rarely visit all states in the first episode, they extremely benefit from action advising based methods, especially for our proposed methods. As L increases to 100, the student has much less chance to explore the state space and simply follows the teacher's advised actions in the advised states, resulting in lower Last and Overall Average performance. Similar results can be seen in AdhocTD-Decay with different λ. When λ=0.99, the student rarely learns by exploration, and finally, the performance is worse than λ=0.9.

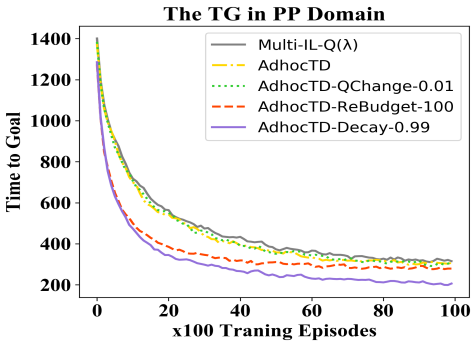
5.2 Predator-Prey domain

Game Description. Predator-Prey (PP) is a grid world domain, which has become the first trial for MAS before applying in more complex situations. In this paper we use the publicly available instantiation of PP domain implemented in [6].[2] In our experiments, there are 5 agents, including 4 predators and 1 prey settled in an $N \times N$ grid world, where N is the number of grids in x (y) direction. Each agent occupies one grid, and chooses between five actions *Stay*, *Go Up*, *Go Down*, *Go Left*, and *Go Right*. By executing an action, they either stay in place or move one grid in one of the four cardinal directions. In this game, only predators are RL agents, while the prey takes a random action 20% of the time, with rest of the time moving away from all predators. The four predators independently observe the environment and learn to catch the prey by cooperation. The prey is caught if four predators are next to the prey in four cardinal directions. In this way, when predators catch the prey, each predator receives a common reward of 1, otherwise 0. Despite this environment's representational and mechanical simplicity, it is still capable of presenting complex cooperative behaviours for MARL. To avoid the challenge of hidden states, each predator has fully observation. At each time step, each predator observes the relative positions of other predators as well as the prey in form of x-axis and y-axis values. All values of states are normalised to $[-1, 1]$ by dividing by the number of grids N. In order to reduce the state space, Tile coding [21] is used to force a generalization with 8 tilings and tile-width 0.5. In all experiments, we use classic RL algorithm, Q(λ), where α=0.1, γ=0.9 and λ=0.9. The ϵ-greedy strategy is used as the exploration strategy for all agents with ϵ=0.1.

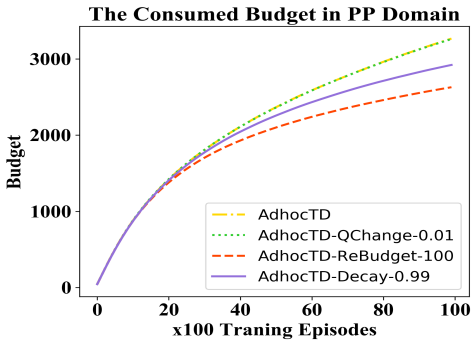
Evaluation Metrics. There is one popular performance metric for PP domain. *Time to Goal* (TG) is the number of steps that predators take to catch the prey. One episode starts when predators and the prey are initialized with a random position in the grid world. The episode ends when either predators catch the prey or a time limit is exceeded. We set the maximum number of steps in each episode as 1,500. The game is trained for 10,000 episodes. After every 100 training episodes, we average the TG values to obtain a more stable value. The process is repeated over 400 runs. The parameters of AdhocTD have been tuned in current task for achieving the best TG values. Then we choose v_a=0.2 and v_b=1 for AdhocTD and AdhocTD-QChange/ReBudget/Decay. To show the effect of different values of parameters, we set: a) η=0, 0.01 and 0.03; b) L=50, 100 and 150; c) ρ=0.9, 0.99 and 0.999.

Experimental Results. Figure 2a shows the performance of TG and the consumption of advising budget, averaged over 400 runs. AdhocTD-Decay with ρ=0.99 has significant lower TG values than all other methods. Since agents in AdhocTD-ReBudget can insist on the advised actions for a longer period, this method has shown to surpass AdhocTD-QChange. Thanks to action advising, the TG of AdhocTD is sightly lower than Multi-IL-Q(λ), while it is still much higher than the TG of AdhocTD-ReBudget/Decay. In Figure 2b, we can see that our proposed methods spend lower budget than AdhocTD. The consumed budget of AdhocTD-Decay is higher than AdhocTD-ReBudget, which may be due to that with L=100, agents in AdhocTD-ReBudget has less chance to ask for advice. The detailed performance of our methods with different parameters is shown in Table 2. We record the first TG value (Initial), the last TG value (Last) and the mean TG value (Mean). As η decreases, agents

[2]The source code is available at http://www.biu-ai.com/RL.



(a) TG of all methods



(b) Budget consuming of all methods

Figure 2: TG and budget consumption of Multi-IL-Q(λ), AdhocTD, and AdhocTD-QChange/ReBudget/Decay when $b_{ask} = b_{give}$=4,000, η=0.01, L=100 and ρ=0.99

are easier to reuse previous advice, and obtain lower TG values. Proper value of η (e.g., η=0.01) helps a student filter some bad advice. Overall, AdhocTD-Decay with ρ=0.99 achieves the lowest last and mean TG values compared with all other methods, which allows agents exploring the state space while reusing advices.

5.3 Half Field Offense

Game Description. Half Field Offense (HFO) is a simulated robot soccer game [14].[3] In our experiments, there are three players and one goalkeeper. Only players are learning from scratch, while the goalkeeper adopts a fixed policy Helios Policy [1]. Each player learns to score a goal when possesses the ball, otherwise move. They choose between four actions *Shoot*, *PassNear*, *PassFar*, and *Dribble*. The players can benefit from cooperative behaviors, for example, one player passes the ball to another player for better shot. In our implementation, a player's state is composed of following observations: 1) whether the player is in possession of the ball; 2) the proximity to the center of the goal; 3) the angle from one player to goal center; 4) the largest angle of one player to the goal without blocking players; 5) the goal opening angle of the nearest (or farthest) partner. These features are normalized in the range $[-1, 1]$ and discretized by Tile Coding with 5 tilings and 0.5 tile

[3]The source code is available at https://github.com/f-leno/AdHoc_AAMAS-17.

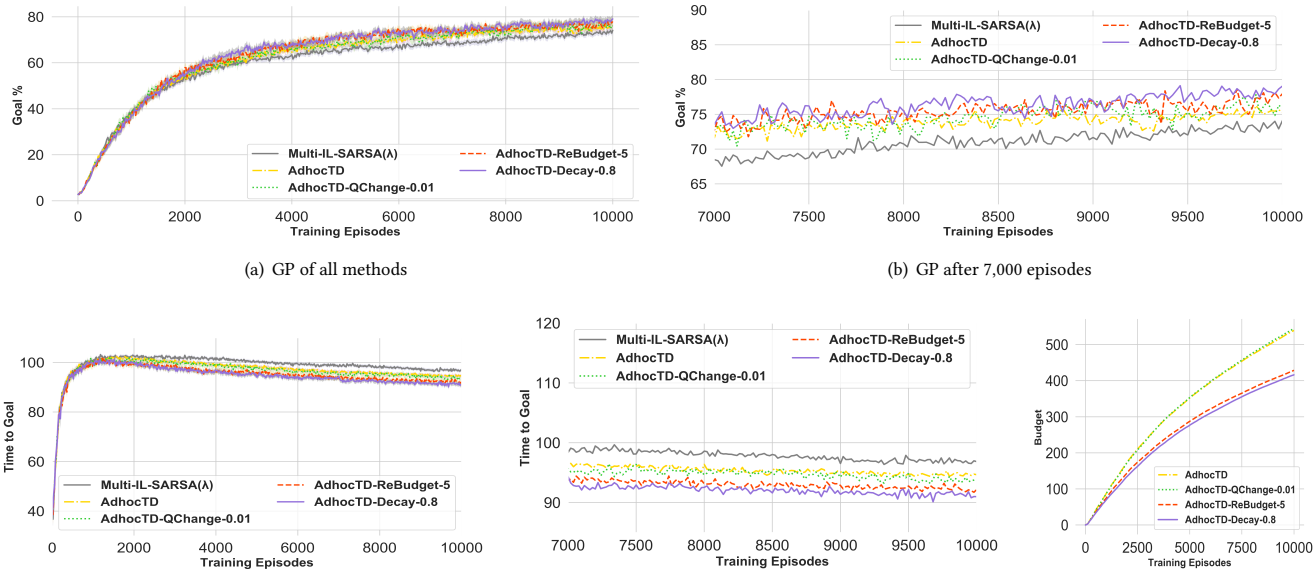
Table 2: Performance metrics for the agents in PP domain (average of over 400 trials). Maximum values of action reusing methods with η, L and ρ are shown in bold respectively. Lower values are better. The best value of each column is underlined.

Agents	Initial	Last	Mean
Multi-IL-Q(λ)	1400.929	314.679	387.757
AdhocTD	1374.108	297.499	371.257
QChange (η=0)	**1380.338**	**297.225**	455.628
QChange (η=0.01)	1384.214	308.214	**449.224**
QChange (η=0.03)	1389.183	307.377	456.549
ReBudget (L=50)	1304.371	311.588	326.361
ReBudget (L=100)	**1274.574**	312.039	**325.001**
ReBudget (L=150)	1276.654	**281.741**	367.076
Decay (ρ=0.9)	1329.096	242.126	323.914
Decay (ρ=0.99)	**1285.165**	**206.067**	**312.541**
Decay (ρ=0.999)	1299.662	239.546	365.294

size. One episode starts when all players and the ball initialized with a random position on the field. The episode ends when either the players scores a goal, the goalkeeper catches the ball, the ball leaves the field, or a time limit is exceeded. When a player scores a goal, all players receive a common reward of 1, otherwise -1. Due to the complexity of this task, we use SARSA(λ) to promote learning speed for all methods. In SARSA(λ), we use α=0.1, γ=0.9, and λ=0.9. The exploration strategy for all agents is ϵ-greedy with ϵ= 0.1.

Evaluation Metrics. We train the players for 10,000 episodes. In each episode, the maximum number of steps is 200. After every 20 training episodes, the learning is halted and 100 testing episodes are played. During testing episodes, all players no longer update their policies, advise each other and reuse advice. They execute currently learned best action in every state. This training process is repeated over 50 runs. We use two standard metrics for performance evaluation. *Goal Percentage* (GP) is the percentage of testing episodes in which a goal is scored. *Time to Goal* (TG) in this game is the average number of steps that the players score a goal during testing episodes. We choose v_a=0.5 and v_b=1.5 for AdhocTD, which is the same as [8]. In order to compare the performance of different parameters, we set: a) η=0.01, 0.05 and 0.1; b) L=5, 10 and 15; c) ρ=0.6, 0.8 and 0.99.

Experimental Results. The average results are shown in Figure 3. We can see that AdhocTD-Decay with ρ=0.8 and AdhocTD-ReBudget with L=5 have much higher GP and lower TG than other methods, while consume less budget. Since agents are still learning, they may advise some non-optimal actions at the early stage of learning. Before about 6,000 episodes, AdhocTD-Decay has better GP than AdhocTD-ReBudget, which means that it is vital for learning students to explore the state space even following their teachers' advices. The first GP/TG value (Initial), the last GP/TG value (Last), the mean GP/TG value (Mean) and the AUC of GP/TG (AUC) are shown in Tables 3-4. Due to the rare chance of exploration in the environment when advising, we can see that AdhocTD-Decay with ρ=0.99 has the lowest GP than all other methods even compared with non-advising method Multi-IL-SARSA(λ).



(a) GP of all methods

(b) GP after 7,000 episodes

(c) TG of all methods

(d) TG after 7,000 episodes

(e) The consumed budget

Figure 3: GP, TG and consumed budget of Multi-IL-SARSA(λ), AdhocTD, AdhocTD-QChange/ReBudget/Decay when b_{ask} = b_{give}=600, η=0.05, L=10 and ρ=0.8

Table 3: Performance metrics regarding GP values in HFO (average of over 50 trials). Higher values are better.

Agents	Initial	Last	Mean	AUC
Multi-IL-SARSA(λ)	2.81	74.1	59.894	29968.255
AdhocTD	3.01	75.31	61.855	30950.24
QChange (η=0.01)	**3.05**	**75.97**	**62.298**	**31171.96**
QChange (η=0.05)	2.81	75.56	62.277	31161.435
QChange (η=0.1)	2.81	72.92	61.787	30917.495
ReBudget (L=5)	3.05	**77.97**	**63.592**	**31819.31**
ReBudget (L=10)	**3.17**	76.74	63.199	31622.525
ReBudget (L=15)	2.65	76.57	62.538	31292.11
Decay (ρ=0.6)	3.0	76.99	63.827	31937.115
Decay (ρ=0.8)	2.71	**79.07**	**63.961**	**32003.74**
Decay (ρ=0.99)	**3.2**	71.47	56.237	28137.575

Table 4: Performance metrics regarding TG values in HFO (average of over 50 trials). Lower values are better.

Agents	Initial	Last	Mean	AUC
Multi-IL-SARSA(λ)	37.64	96.85	98.772	49417.355
AdhocTD	39.5	94.67	96.985	48522.325
QChange (η=0.01)	38.62	93.9	**96.395**	**48227.86**
QChange (η=0.05)	**35.19**	**93.47**	96.403	48233.77
QChange (η=0.1)	37.82	93.84	97.194	48628.42
ReBudget (L=5)	40.85	92.17	94.829	47442.88
ReBudget (L=10)	38.38	**91.94**	94.59	47324.67
ReBudget (L=15)	**37.71**	92.4	**94.4**	**47229.565**
Decay (ρ=0.6)	38.1	91.59	94.675	47367.405
Decay (ρ=0.8)	**37.85**	**90.99**	**94.289**	**47174.4**
Decay (ρ=0.99)	39.5	94.0	95.464	47760.86

6 CONCLUSION AND FURTHER WORKS

In this paper, we address how to use advice more effectively in the teacher-student paradigm, which are particularly suitable for budget constraint. Specifically, with our proposed methods, when an agent encounters a state advised before, it either reuses previous advice, asks for a new advice or select the next action in a usual exploration strategy. Our main finding is that reusing advice achieves significantly better performance than the advising framework without specifying advice reusing. Moreover, when all agents are learning and adapting their policies, it is vital for them to explore while reusing the advised actions. Last but not least, insisting on non-optimal teachers' advice for too long may even hinder the overall learning process.

As future work, we consider to filter the influence of bad advice, and learn a model to represent the shared knowledge from teachers. A student can also learn how to choose reusing advice, following the exploration strategy or asking for advice. Such decision can be included into the high-level advising framework like LeCTR [17].

ACKNOWLEDGEMENT

This work was supported by the Fundamental Research Funds for the Central Universities, SCUT (No. 2017ZD048, D2182480), the Science and Technology Planning Project of Guangdong Province (No. 2017B050506004), the Science and Technology Programs of Guangzhou (No. 201704030076, 201802010027, 201902010046).

REFERENCES

[1] Hidehisa Akiyama. 2012. Helios team base code. (2012). https://osdn.jp/projects/rctools/

[2] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara J. Grosz. 2016. Interactive teaching strategies for agent training. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*. 804–811.

[3] Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking Fast and Slow with Deep Learning and Tree Search. In *Advances in Neural Information Processing Systems*. 5360–5370.

[4] Brenna D. Argall, Sonia Chernova, Manuela M. Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57, 5 (2009), 469–483.

[5] Andrew G. Barto., Philip S. Thomas, and Richard S. Sutton. 2017. Some Recent Applications of Reinforcement Learning. In *Proceedings of the 18th Yale Workshop on Adaptive and Learning Systems*.

[6] Tim Brys, Ann Nowé, Daniel Kudenko, and Matthew Taylor. 2014. Combining Multiple Correlated Reward and Shaping Signals by Measuring Confidence. In *Proceedings of 28th AAAI Conference on Artificial Intelligence*. 1687–1693.

[7] Jeffery A. Clouse. 1996. *On integrating apprentice learning and reinforcement learning*. PhD thesis, University of Massachusetts.

[8] Felipe Leno da Silva, Ruben Glatt, and Anna Helena Reali Costa. 2017. Simultaneously Learning and Advising in Multiagent Reinforcement Learning. In *Proceedings of the 16th International Conference on Autonomous Agents and MultiAgent Systems*. 1100–1108.

[9] Anestis Fachantidis, Matthew E. Taylor, and Ioannis P. Vlahavas. 2017. Learning to Teach Reinforcement Learning Agents. *Machine Learning and Knowledge Extraction* 1 (2017), 21–42.

[10] Anna Helena Reali Costa Felipe Leno da Silva. 2019. A Survey on Transfer Learning for Multiagent Reinforcement Learning Systems. *Journal of Artificial Intelligence Research* 64 (2019), 645–703.

[11] Vaibhav Gupta, Daksh Anand, Praveen Paruchuri, and Balaraman Ravindran. 2019. Advice Replay Approach for Richer Knowledge Transfer in Teacher Student Framework. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 1997–1999.

[12] Ercüment Ilhan, Jeremy Gow, and Diego Perez Liebana. 2019. Teaching on a Budget in Multi-Agent Deep Reinforcement Learning. *ArXiv* abs/1905.01357 (2019).

[13] Dong-Ki Kim, Miao Liu, Shayegan Omidshafiei, Sebastian Lopez-Cot, Matthew Riemer, Golnaz Habibi, Gerald Tesauro, Sami Mourad, Murray Campbell, and Jonathan P. How. 2019. Learning Hierarchical Teaching in Cooperative Multiagent Reinforcement Learning. *ArXiv* (2019), abs/1903.03216.

[14] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. 1997. RoboCup: A Challenge Problem for AI. *AI Magazine* 18 (1997), 73–85.

[15] Jens Kober, J. Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal Of Robotics Research* 32 (2013), 1238–1274.

[16] Laëtitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. 2012. Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. *Knowledge Eng. Review* 27 (2012), 1–31.

[17] Shayegan Omidshafiei, Dong-Ki Kim, Miao Liu, Gerald Tesauro, Matthew Riemer, Christopher Amato, Murray Campbell, and Jonathan P. How. 2019. Learning to Teach in Cooperative Multiagent Reinforcement Learning. In *The Thirty-Third AAAI Conference on Artificial Intelligence*. 6128–6136.

[18] IEEE Transactions on Computational Intelligence and AI in Games. 2012. The Mario AI Benchmark and Competitions. 4, 1 (2012), 55–67.

[19] Bob Price and Craig Boutilier. 2003. Accelerating Reinforcement Learning through Implicit Imitation. *Journal of Artificial Intelligence Research (JAIR)* 19 (2003), 569–629.

[20] G. A. Rummery and M. Niranjan. 1994. *On-line Q-learning using connectionist systems*. Technical Report CUED/F-INFENG/TR 166. Cambridge University Engineering Dept.

[21] Alexander A. Sherstov and Peter Stone. 2005. Function Approximation via Tile Coding: Automating Parameter Choice. In *Proc. Symposium on Abstraction, Reformulation, and Approximation (SARA-05)*. Edinburgh, Scotland, UK.

[22] Halit Bener Suay, Tim Brys, Matthew E. Taylor, and Sonia Chernova. 2016. Learning from Demonstration for Shaping through Inverse Reinforcement Learning. In *Proceedings of the 2016 International Conference on Autonomous Agents Multiagent Systems*. 429–437.

[23] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction* (1nd. ed.). MIT press, Cambridge, MA, USA.

[24] Ming Tan. 1993. Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents. In *10th International Conference on Machine Learning*. 330–337.

[25] Lisa Torrey and Matthew E Taylor. 2012. Help an Agent Out: Student/Teacher Learning in Sequential Decision Tasks. In *Proceedings of the 2012 Adaptive and Learning Agents Workshop (AAMAS)*. 41–48.

[26] Lisa Torrey and Matthew E. Taylor. 2013. Teaching on a budget: agents advising agents in reinforcement learning. In *Proceedings of 12th the International Conference on Autonomous Agents and MultiAgent Systems*. 1053–1060.

[27] Zhaodong Wang and Matthew E. Taylor. 2017. Improving Reinforcement Learning with Confidence-Based Demonstrations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*. 3027–3033.

[28] Christopher J.C.H. Watkins and Peter Dayan. 1992. Technical Note: Q-learning. *Machine Learning* 8 (1992), 279–292.

[29] Changxi Zhu, Ho fung Leung, Shuyue Hu, and Yi Cai. 2019. A Q-values Sharing Framework for Multiple Independent Q-learners. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2324–2326.

[30] Matthieu Zimmer, Paolo Viappiani, and Paul Weng. 2014. Teacher-Student Framework: A Reinforcement Learning Approach. In *AAMAS Workshop Autonomous Robots Multirobot Systems*.