Agents Teaching Humans in Reinforcement Learning Tasks

Yusen Zhan[1], Anestis Fachantidis[2], Ioannis Vlahavas[2], Matthew E. Taylor[1]
[1]Washington State University, Pullman, WA, USA
[2]Aristotle University of Thessaloniki, Thessaloniki, Greece
yusen.zhan@wsu.edu, {afa, vlahavas}@csd.auth.gr, taylorm@eecs.wsu.edu

ABSTRACT

This paper extends our existing teacher-student framework to allow a knowledgeable agent to teach human students. An agent teacher instructs a human student by suggesting actions the student should take as it learns. This paper extends previous algorithms, used for agents teaching other agents, to develop several new algorithms for agents teaching humans. Our results in the Pac-Man domain show that our new approaches can indeed be effectively used to improve human learning. Moreover, some of these human-teaching approaches perform better than some of the original algorithms when one agent teaches another agent.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Miscellaneous

General Terms

Algorithms, Experimentation

Keywords

Reinforcement Learning, Inter-agent teaching, Human Teaching

1. INTRODUCTION

Agents can autonomously learn to master sequential decision tasks by reinforcement learning (RL) [8]. Traditionally, RL agents are trained and used in isolation. Research focusing on interaction among agents, even between agents and humans, becoming popular.

This paper investigates how an RL agent could serve as a teacher for human students. This limits us to human-understandable teaching methods, prevents teachers from assuming any access to a student's internal representation, and prevents students from simply starting with the teacher's knowledge. Furthermore, it requires teachers to be able to instruct students who may learn and perceive their environment differently.

As a motivating example of RL agents used as teachers, consider the fast-growing industry of computer games. Modern games often have built-in training sessions to help them; currently, this is additional content created by game developers. However, most of these sessions are simple built-in processes without any intelligent guidance. RL agents could learn to play these games autonomously and then teach human players, reducing the amount of developer time required to produce training content and providing better player game experience.

There are many possible methods to assist agent's learning [9, 10], but few are also applicable to human students. One that is applicable is *action advice*: as the student practices, the teacher suggests actions to take. This method is the one we follow in this work and it requires only agreement of the action sets between teachers and students, while allowing for different state representations and different learning algorithms among teachers and students.

Another assumption we make in this paper is that the agent teachers can only give their students a limited amount of advice. The primary reason for this restriction is that human students would have limited patience and attention. In some domains, the communication between agents is also limited. Moreover, a teacher that over-advises a student could actually hinder its learning, if the differences between them are significant or the teacher is sub-optimal.

This paper extends our previous work [10], which focused on agents teaching agents, by studying how an RL agent can best teach human students using a limited advice budget. This paper will also evaluate these algorithms on cases when one agent teaches another agent. The teacher observers states and actions of student and gives advice a fixed number of times, when necessary, but it cannot observe or change internal information of the student. The advice necessity is defined in a more formal way later in this text.

One unique problem with human students is that they cannot process very frequent advice from an agent because an agent has shorter reaction time than that of a human. To address this problem we extend the previous teaching algorithms to 1) reduce the total amount of advice given to a human student and 2) not provide advice too often. Both modifications are designed to improve the user's experience while maintaining teaching effectiveness.

We evaluate these algorithms experimentally in Pac-Man. The results show that our new approaches can not only be applied to agents teaching humans, but also to agents teaching agents settings. Moreover, some of our new approaches perform better than the old ones in the agents teaching agents setting, even though this was not the objective of our algorithmic design. Furthermore, this paper proposes a new theoretical formulation of the advising problem representing each of the old and new algorithms as parameterized instances of a more general *advice distribution policy.*

2. BACKGROUND AND RELATED WORK

2.1 Reinforcement Learning

In reinforcement learning, an agent learns through trial and error to perform a task in an environment. As the agent takes actions, it receives feedback in the form of real-valued rewards. RL algorithms use this information to gradually improve an agent's control policy in order to maximize its total long-term expected reward.

At each step, the agent observes the state s of its environment. Using its policy π, it selects and performs an action a, which alters the environment state to s'. The agent observes this new state as well as a reward r, and it uses this information to update its policy. This cycle repeats throughout the learning process, which is often broken into a sequence of independent episodes.

A common way to represent a policy is with a Q-function $Q(s,a)$, which estimates the total reward an agent will earn starting by taking action a in state s. Given an accurate Q-function, the agent can maximize its rewards by choosing the action with the maximum Q-value in each state. Learning a policy therefore means updating the Q-function to make it more accurate. Even in the early stages of learning, the agent chooses actions with maximum Q-values most of the time, but to account for potential inaccuracies in the Q-function, it must perform occasional exploratory actions. A common strategy is ϵ-greedy exploration, where with a small probability ϵ, the agent chooses a random action.

In an environment with a reasonably small number of states, the Q-function can simply be a table of values with one entry for each state-action pair. Basic RL algorithms make updates to individual Q-value entries in this table. However, in some larger environments, states cannot be enumerated and need to be described by features $\{f_1, f_2, ...\}$. The Q-function is then an approximation, and a common form is a linear function $Q(s,a) = \sum_i w_i f_i$. Learning a policy then means updating the weights $\{w_1, w_2, ...\}$.

For the experiments in this paper, we use Sarsa(λ) with linear Q-function approximation [8]. Since they already allow for off-policy exploratory actions, the algorithm can simply treat advice like a particularly lucky form of exploration. These algorithms have four parameters: the exploration rate ϵ, the learning rate α, the eligibility-trace parameter λ, and the discount factor γ. We report their values in each task for reproducibility but we omit a detailed discussion of them.

The weights of the approximated Q-function, $\{w_1, w_2, ...\}$ need to be given initial values. The usual choices are *optimistic*, so that weights are adjusted downwards over time, or *pessimistic*, so they are adjusted upwards. We find that this choice is important in the context of teaching with advice. With optimistic initialization, agents focus their attention on unexplored actions, which means that they delay repeating advised actions. With pessimistic initialization, agents have no such bias and can benefit much more from advice. All agent-learning experiments in this paper therefore use pessimistic initialization.

2.2 Related Work

There are several types of related work in the area of helping to learn. Some of this work focuses on teaching in non-RL settings [3, 7].

In the field of *transfer learning* in RL [9], an agent uses knowledge from a source task to aid its learning in a target task. However, agents perform transfer knowledge from one task to another.

More closely related work has one RL agent teach another without a direct knowledge transfer. For example, in *advice teaching* [10], a student receives advice from a trained and excellent teacher. Our work is mainly based on this work. However, the student may be not only a agent, but also may be a human student in our settings.Other examples include *imitation learning* [5], *apprentice learning* [4].

Finally, humans are sometimes employed to teach agents. For instance *Learning from Demonstration* [1] includes a broad category of work that focuses on agents learning to mimic a human demonstrator. This type of work is the dual of our work — we employ agents to teach humans.

3. TEACHING ON BUDGET

Suppose that an RL agent has learned an effective policy π for a given task ω. Using this fixed policy, it will teach students beginning to learn the same task. As the human/agent student learns, the teacher will observe each state s the student encounters and each action a the student takes. Having a budget of B advice, the teacher can choose to advise the student in $n \leq B$ of these states to take the "correct" action $\pi(s)$. In this work, we assume the teacher's advice is always correct.

How should the teacher spend its n advice most effectively? The teacher should adopt some form of *advice distribution policy* which will guide it to decide *when* to give advice. When students are humans, this distribution policy of the advice should also take into consideration factors such as the possible annoyance from repetitive advising. We take an experimental approach to answer this question, proposing and testing several algorithms that distribute the budget of advice, based on our previous work [10]. We first begin by proposing a new, more general formulation of the advising problem which aims to be flexible enough to cover both past and possibly future methods.

Assume that a RL teacher agent T is trained in a task ω and has access to its learned Q-Value function Q_τ. Then, a student agent S, either human or agent, begins training in ω and is able to accept advice in the form of actions from the teacher. We further assume that the teacher has a limitation on the amount of advice he can give, a budget of B.

We consider two things as the important when advising: the quality of the given advice in the advice *production* phase, and the distribution of the advice in the *distribution* phase, meaning the spread of the advice given in an episode or a decision of giving less advice than that available in the budget B.

For the teacher to **produce** advice we can extract a policy $\pi_\tau = \arg\max_a(Q_\tau(s,a))$ from its action-value function and advice according to it. Note that we assume that the teacher's advice is always *near* optimal and that we greedily exploit its value function.

For the teacher to **distribute** its advice most effectively we define an *advice distribution policy*, π_d as the policy:

$$\pi_d(S, B, I(\cdot), \rho(\cdot)) \rightarrow \{\pi_t, \emptyset\} \tag{1}$$

which distributes the advice with respect to the following: i) the current knowledge about the student S which could be his current state, the action he is intended to take and can possibly be mistaken, or the reward he received; ii) the available budget of advice B; iii) an importance function I, capable of calculating the importance of a given state; and iv) an advice rate function ρ which calculates the rate at

Algorithm 1 Alternate Advising

```
procedure $AlternateAdvising(\pi, n, m)$
1: $step \leftarrow 0$
2: for each student state $s$ do
3:     if $n > 0$ and $step \mod m = 0$ then
4:         $n \leftarrow n - 1$
5:         Advise $\pi(s)$
6:     end if
7:     $step \leftarrow step + 1$
8: end for
end procedure
```

Algorithm 2 Importance Advising

```
procedure ImportanceAdvising$(\pi, n, t)$
1: for each student state $s$ do
2:     if $n > 0$ and $I(s) \geq t$ then
3:         $n \leftarrow n - 1$
4:         Advise $\pi(s)$
5:     end if
6: end for
end procedure
```

which the advice is given (the advice sparsity). Given the previous, an *advice distribution policy* returns either the action suggested by the teacher's policy, $\pi_t(s)$ for the student's current state s or the empty set, $\emptyset$ representing the teacher's *no advice* response at the current step.

The $I(\cdot)$ function governs the *qualitative* characteristics of advising such as the importance of a state or even the lack of attention of a human student (if one has access to such information). The function $\rho(\cdot)$ governs the *quantitative* aspects of advising, such as advising in a steady constant rate of steps m, or a decaying rate of advising. Functions I and ρ along with their parameters (e.g., thresholds), impose their qualitative and quantitative constraints to an advising problem that would else be unconstrained giving by default all the advice immediately and in whatever states.

As a minimal example of the proposed formulation, the Early Advising method [10] which gives all the available advice B immediately, in the first B steps, can be said to follow an advice distribution policy $\pi_d(S = \{s\}, B, I = \emptyset, \rho = 1)$. It requires knowledge only about the current state, s of the student, $S = \{s\}$, it has no importance metric, $I = \emptyset$ and a constant rate function $\rho = 1$, giving advice to every step until the entire budget is used.

3.1 Alternating Advice

In our previous paper [2], an advising method called *Alternating Advice* was proposed. This method is based on the intuition that students should benefit more from advice early on, when they know very little. It also addressed the shortcomings of an advice budget being spent too quickly in a limited part of the state space. In this method, the teacher gives advice once every m steps for the first $n \times m$ states the student encounters, where n equals the budget B of available advice, $n = B$. Based on the previously introduced notion of the advice distribution policy (see Equation 1) this method follows a policy, π_d of the form: $\pi_d(\{s\}, n, \emptyset, m)$. We will adopt this method as a baseline, as it outperforms the Early Advising Algorithm. See Algorithm 1 for details.

3.2 Importance Advising

When all states in a task are equally important to achieve the goal, Early Advising and Alternate Advising could be effective strategies. However, we hypothesize that in some tasks, some states are more important than others, and saving advice for more important states would be a more effective strategy. Therefore, *Importance Advising*[10] . determines when the teacher should provide advice to the student. A threshold measures state importance. The teacher could thus give advice only when the importance of the student's state reaches some threshold t. Algorithm 2 shows the original Importance Advising algorithm.

Recall that a Q-value, $Q(s, a)$, is an estimate of the return for taking action a in state s. If the Q-values for all the actions in s are the same, it does not matter which one is taken, and the state s is unimportant. However, if some actions in s have larger Q-values than others, then it does matter, and s has some "importance." For this paper, we define state importance as:

$$I(s) = \max_a Q(s, a) - \min_a Q(s, a)$$

introduced by Clouse [4] in his work on apprenticeship learning, It was used there to approximate a learner's confidence in a state. Here, we compute $I(s)$ from the teacher's fully-learned Q-function rather than the student's partially-learned one, and in this context it is considered an indicator of state importance rather than of agent confidence. According to our formulation, importance advising can be said to follow an advice distribution policy of the form $\pi_d(\{s\}, n, I(s), 1)$.

However, when teaching humans, the original Importance Advising may provide advice frequently which will detect from the player's experience during training. For example, the human players will continually receive advice from the agent teacher when the ghosts are approaching Pac-Man because these states are important in the sense of our measure. Hence, this paper develops teaching strategies that not only provide the advice to human players but also aim to maintain the player's enjoyment.

The first method we propose to address this problem, introduces a dynamic threshold t' which will vary at runtime. The basic idea is that the advice threshold should be significantly increased after each advising. Therefore, the probability that the teacher will give advice to the student becomes very small due to increased threshold. Then, on every step, the threshold gradually decreases over time, leading again to a higher probability of giving advice which guarantees that the teacher will still give necessary advice to its human students. This intuition implements an advice distribution policy of the form: $\pi_d(\{s\}, n, I(s, t'), 1)$.

Suppose we have a threshold t and a factor $f \geq 1$ that will produce a larger threshold $t' = f \times t$ after every teacher advising. For the threshold t' to be dynamic, a discount factor $\beta \in [0, 1]$ is introduced. We update the threshold t' as $t' = \beta \times t$ in each step so that t' gradually decreases over time. Finally, if the dynamic threshold t' is less than threshold t, set $t' = t$. If t' becomes too small, the teacher would save advice for unimportant states. The dynamic threshold t' vary from $f \times t$ to t. After advising, the threshold is set again to $t' = f \times t$. We call this method *Dynamic Importance Advising* (see Algorithm 3).

Lines 1, 6 and 8-10 are new in Algorithm 3, compared to Algorithm 2. Line 1 introduces the dynamic threshold

Algorithm 3 Dynamic Importance Advising

```
procedure DynamicImportanceAdvising($\pi, n, t, f, \beta$)
 1: $t' \leftarrow t \times f$
 2: for each student state $s$ do
 3:   if $n > 0$ and $I(s) \geq t'$ then
 4:     $n \leftarrow n - 1$
 5:     Advise $\pi(s)$
 6:     $t' \leftarrow t \times f$
 7:   end if
 8:   $t' \leftarrow \beta \times t'$
 9:   if $t' < t$ then
10:     $t' \leftarrow t$
11:   end if
12: end for
end procedure
```

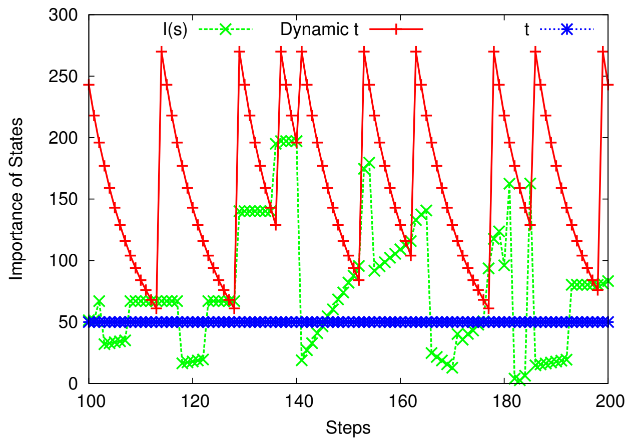


Figure 1: Comparison of the current state's importance ($I(s)$), the dynamic threshold of Dynamic Importance Advising (Dynamic t) and Important Advising (t) in part of one episode.

and Line 6 resets the dynamic threshold. Finally, Lines 8-10 guarantee that the threshold will vary over time with a lower bound. This method is a generalization of Importance Advising. When $f = 1$, Dynamic Importance Advising is equivalent to Importance Advising. Figure 1 illustrates the difference between Dynamic Importance Advising and Importance Advising in one example run of the algorithm when one knowledgeable agent teaches a naive student agent.

Figure 1 shows that our new approach significantly reduces the amount of advice given since there are a few red points (dynamic threshold of state importance) that meet the green points (actual state importance on time step t). In contrast, a lot of green points are above the blue point, indicating that the agent teacher will keep sending advice to the student. Given that, we can say that Dynamic Importance Advising reduces the giving rate of advice indirectly through the varying importance of the states.

The second method proposed here to improve importance advising implements the intuition that no advice should be given for m steps after any advising event. Such a method reduces directly the advice rate by introducing a constant advice rate function $\rho = m$. We call this algorithm *Alternative Importance Advising*. See Algorithm 4 for details.

Algorithm 4 Alternative Importance Advising

```
procedure AlternativeImportanceAdvising($\pi, n, m, t$)
 1: $step \leftarrow m$
 2: for each student state $s$ do
 3:   if $step = 0$ then
 4:     if $n > 0$ and $I(s) \geq t$ then
 5:       $n \leftarrow n - 1$
 6:       Advise $\pi(s)$
 7:       $step \leftarrow m$
 8:     end if
 9:   else
10:     $step \leftarrow step - 1$
11:   end if
12: end for
end procedure
```

Algorithm 5 Mistake Correcting

```
procedure MistakeCorrecting($\pi, n, t$)
 1: for each student state $s$ do
 2:   Observe student's announced action $a$
 3:   if $n > 0$ and $I(s) \geq t$ and $a \neq \pi(s)$ then
 4:     $n \leftarrow n - 1$
 5:     Advise $\pi(s)$
 6:   end if
 7: end for
end procedure
```

This algorithm implements an advice distribution policy: $\pi_d(\{s\}, n, I(s), m)$. Note that if $m = 1$, this method is also equivalent to the original Importance Advising algorithm. Lines 1, 8 and 11 are new in Algorithm 4, compared to Algorithm 2.

3.3 Mistake Correcting

Even if a teacher saves its advice for important states, it may end up wasting some advice in states where the student already intended to take the correct action. Based on this observation, the *Mistake Correcting* algorithm was proposed in previous work [10] (see Algorithm 5). Since Mistake Correcting assumes knowledge about the student's intended action a (not only his state) it can be said to follow an advice distribution policy of the form: $\pi_d(\{s, a\}, n, I(s), 1)$.

In order to address the frequent advice problem in Mistake Correcting, we introduce the dynamic threshold t' to Mistake Correcting, to get Dynamic Mistake Correcting, which will also reduce advice frequency during the runtime. Lines 1, 7 and 9-12 are new in Algorithm 6, compared to Algorithm 5. Line 1 introduces the dynamic threshold and Line 7 resets the dynamic threshold. Finally, Lines 9-12 guarantee that the threshold varies over time with a lower bound.

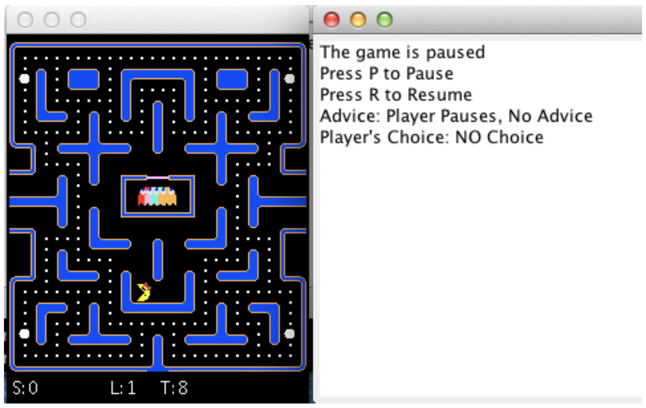
Just as in standard Mistake Correcting, this method requires that the student will announce its intended action in advance and give teachers an opportunity to correct them, since one of the key assumptions in this work is that teachers have no direct access to the student's knowledge.

Firstly, We also introduce dynamic threshold to Mistake Correcting. See Algorithm 6 for details.

When t is 0, this approach ignores state importance and corrects all mistakes. When $f = 1$, it becomes equivalent to the original mistake correcting method. The second method is the same as Algorithm 4 in which there is no advice for

Algorithm 6 Dynamic Mistake Correcting

```
procedure $DynamicMistakeCorrecting(\pi, n, t, f, \beta)$
1:  $t' \leftarrow t \times f$
2:  for each student state $s$ do
3:      Observe student's announced action $a$
4:      if $n > 0$ and $I(s) \geq t'$ and $a \neq \pi(s)$ then
5:          $n \leftarrow n - 1$
6:          Advise $\pi(s)$
7:          $t' \leftarrow f \times t$
8:      end if
9:      $t' \leftarrow \beta \times t'$
10:     if $t' < t$ then
11:         $t' \leftarrow t$
12:     end if
13: end for
end procedure
```



The game is paused
Press P to Pause
Press R to Resume
Advice: Player Pauses, No Advice
Player's Choice: NO Choice

Figure 2: Pac-Man with Information Display for Human Players

m steps after advising.

Due to the space, we omit the details of Alternative Mistake Correcting. If $m = 1$, the Alternative Mistake Correcting is equivalent to the original Mistake Correcting method. Finally, Algorithms 1d **??** can be said to follow advice distribution policies of the form $\pi_d(\{s, a\}, n, I(s, t'), 1)$ and $\pi_d(\{s, a\}, n, I(s), m)$ respectively.

4. EXPERIMENTAL DOMAIN AND RESULTS

This section first introduces the experimental domain. Then, it presents the results of agents teaching agents, demonstrating that although our algorithms were designed for teaching human students, they also work for agents teaching agents. Finally, the results of agents teaching humans are presented.

4.1 Pac-Man

Pac-Man is a famous 1980s arcade game in which the player navigates a maze like the one in Figure 2, trying to earn points by touching edible items and trying to avoid being caught by the four ghosts. We use a JAVA implementation of the game provided by the Ms. Pac-Man vs. Ghosts League [6], which conducts annual competitions. Ghosts in our setting are the standard ones which will chase the player 80% of the time and choose the actions randomly 20%.

Pac-Man episodes all occur in mazes. For agents teaching agents, we use the default maze. To avoid systematic biases, we use a new maze for human players. The player and all ghosts have four actions — move up, down, left, and right — but some actions are unavailable due to the walls in the maze. Four moves are required to travel between the small dots on the grid, which represent food pellets and are worth 10 points each. The larger dots are power pellets, which are worth 50 points each. When the player gets the lager ones, the ghosts become edible for a short time, during which they slow down and turn to fleeing instead of chasing. Eating a ghost is worth 200 points and respawns the ghost in the lair at the center of the maze. The episode ends if any ghost catches Pac-Man, or after 2000 steps in the agents teaching agents setting.

This domain is discrete but has a very large state space. There are over 1000 distinct locations in the maze, and a complete state consists of the locations of Pac-Man, the ghosts, the food pellets, and the power pills, along with each ghost's previous move and whether or not it is edible. The combinatorial explosion of possible states makes it essential to approach this domain through high-level feature construction and Q-function approximation.

In this paper, we follow our previous work that adopted the action-specific features as a high-level feature set. When using action-specific features, a feature set is really a set of functions $\{f_1(s, a), f_2(s, a), ...\}$. All actions share one Q-function, which associates a weight with each feature. A Q-value is $Q(s, a) = w_0 + \sum_i w_i f_i(s, a)$. To achieve gradient-descent convergence, it is important to have the extra bias weight w_0 and to also normalize the features to the range $[0, 1]$.

In this paper, we create agents with a state representation in Pac-Man domain by defining a feature set, which consists of 7 features that count objects at a range of distances from Pac-Man maze, as we used (and defined) in our previous work [10].

A perfect score in an episode is 5600 points, but this is quite difficult to achieve (for both human and agent players). An agent executing random actions earns an average of 250 points. The agent can learn to catch more edible ghosts and achieve a average of 3800 points per episode.

For agents teaching Pac-Man to humans, there are some problems which need to be addressed. For example, how should the teacher advice be displayed? Human players cannot instantly accept the advice immediately — how should time be provided to human players to learn? Moreover, how do the human players accept the advice from the teacher? To tackle all above issues, we extend the GUI of the original Pac-Man implementation to provide this information (see Figure 2).

When a human is playing as the student, s/he can play the game as normal, navigating through the maze using the keyboard. When the agent teacher wants to give the advice to the human student, the agent teacher pauses the game and displays the advised action — up, down, left, or right — in the message frame. To teach the human students, we develop a user-choice mode to give the advice in which human players are able to choose the actions. Since the game is paused, the human student has sufficient time to analyze the current situation and decide whether or not to accept the advice from the teacher. It is also important for the advice to be considered as teaching and learning process rather that an instantaneous correcting behaviors of players

#	Advising Method	Avg. Reward
1	Mistake Correcting	3387
2	Alternative Mistake Correcting	3299
3	Dynamic Mistake Correcting	3249
4	Importance Advising	3192.4
5	Dynamic Importance Advising	3138
6	No Advice	3100
7	Alternative Importance Advising	3084.5

Table 1: Average reward of 1000 episodes for all the methods used in the agents teaching agents setting. The results are averaged over 12 independent trials.

during the pausing. After choosing an action, the human student can resume the game by pressing R button.

4.2 Agents Teaching Agents

This subsection presents results from agents teaching agents using our new teaching algorithms. To compare with previous algorithms in [10], we adopt the same experimental settings and statistical methods. The student agent must accept the advice action from the teacher.

To smooth the natural variance in student performance, each learning curve averages at least 15 independent trials of student learning. While training, an agent pauses every few episodes to perform at least 30 evaluation episodes and record its average performance. No learning, exploration, or teaching takes place during evaluation episodes. This way the learning curves reflect only the student's knowledge, not teacher knowledge.

Since we have many parameters (i.e., t, f, and β) in our new approaches, we used preliminary experiments to chose the best — we report the results only for tuned parameters.

Each Pac-Man teacher is given an advice budget of $n = 1000$, which is exact half the number of the steps limit in a single episode. The RL parameters of the agents are $\epsilon = 0.05$, $\alpha = 0.001$, $\gamma = 0.999$, and $\lambda = 0.9$.

We present experiments where a trained Sarsa teacher provides advice to a Sarsa student (Figure 3, left). We also compare to the performance of previous algorithms: Mistake Correcting and Importance Advising. The parameter details are as follows: Alternative Importance Advising, $m = 30$ and $t = 150$; Alternative Mistake Correcting, $m = 15$ and $t = 210$;Dynamic Importance Advising, $t = 40$, $f = 25$ and $\beta = 0.9$; Dynamic Mistake Correcting, $t = 20$, $f = 25$ and $\beta = 0.9$; Importance Advising, $t = 50$; Mistake Correcting, $t = 50$. The average score through all 1000 episodes shows that alternative mistake correcting is the second best advising method from all seven tested, with standard mistake correcting being the best (see Table 1).

These new approaches aim to reduce the amount of advice given in the early episodes, while not significantly reducing the performance. The results from Figure 3 (left) confirms this, but the best performing algorithm is Mistake Correcting methods which indicates that they provide advice in the most critical points, improving relative to no advice in previous work [10].

4.3 Agents Teaching Humans

In agents teaching humans setting, Pac-Man teachers are given an advice budget of $n = 1000$ as well. We designed four experimental groups with a total of forty human players. Each group has ten players and conducted three rounds during the experiments. After the players completed the experiments, the player received compensation (\$0, \$0.5, or \$1) based on their scores in Round 3. After round three, the players completed a survey. The four groups are designed as follows:

Control Group—There is no advising for human players during the experiments.

Baseline Group—The Importance Advising is adopted as the advising algorithm for teaching human players. The parameter that the agent teacher uses is $m = 30$.

Dynamic Group—The Dynamic Importance Advising algorithm is adopted as the advising algorithm for teaching human players. The parameters that the agent teacher uses are $t = 50$, $f = 100$ and $\beta = 0.9$.

Alternative Group—The Alternative Mistake Correcting algorithm is adopted as the advising algorithm for teaching human players. The parameters that the agent teacher uses are $m = 20$ and $t = 100$.

These parameters are chosen by preliminary experiments among small group WSU students before the case study. The details of three rounds are designed as follows:

Round 1: There is a 1000 step limit. After 1000 steps, regardless of the game's state, the game will be terminated. If the player dies in less than 1000 steps, the players will start new games until s/he has taken 1000 steps.

Round 2: There is again a 1000 step limit, but now the player will receive advice (except for the Control Group). Again, if the player dies in less than 1000 steps, the players will start new games until s/he has taken 1000 steps.

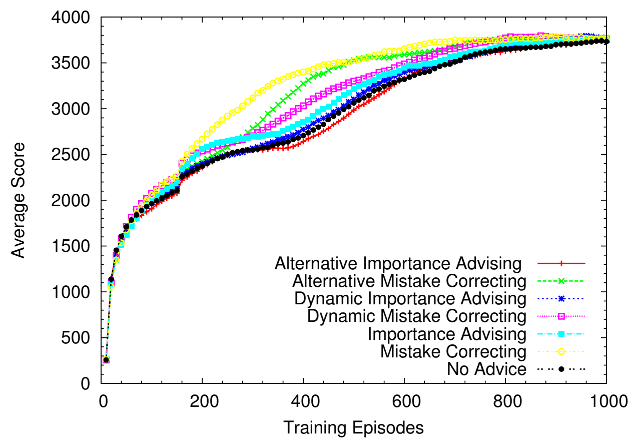
Round 3: In the third and final round, the players have only one chance to play the game. If they die or complete the game (i.e., eat all pellets), the game will be terminated.

In Figure 4, the left figure presents the average scores that human players obtain during the whole experiment and the right figure renders the median scores that human players obtained during the whole experiment. To offer these two statistical estimations, we want to give a more comprehensive performance results in each group. "Control" denotes the Control Group, "Dynamic" denotes the Dynamic Group, "Baseline" denotes the Baseline Group, and "Alternative" denotes the Alternative Group.

In Figure 4, we can see that the Alternative Group performed better than the others two groups with advice (Baseline Group and Dynamic Group), compared to the Control Group. In Round 1, the difference between groups is not significant. In Round 2, the scores range from 2000 to 2400; The range is from 1700 to 2700 in Round 3. In Figure 4 right we have similar results except an unexpected point for Baseline Group in Round 1, which might be explained as an adaptation phase for most of players in Baseline Group[1], since they improve their performance in Round 2 and finally achieve a higher median score.

Note that the setting of Round 3 is more strict than Round 1 and Round 2 because the players have only one chance to play the game — if the players immediately make a fatal mistake, the game will be terminated. Thus the scores between Round 3 and the other rounds cannot be directly compared. In Round 3, all groups except for Alternative Group gain

[1]Figure 4 (right) contains median scores that human players finally obtain in different groups. If the median score is lower than the average score for a specific group, that means most of players in this group perform poorly and obtain lower scores which are smaller than the average score.



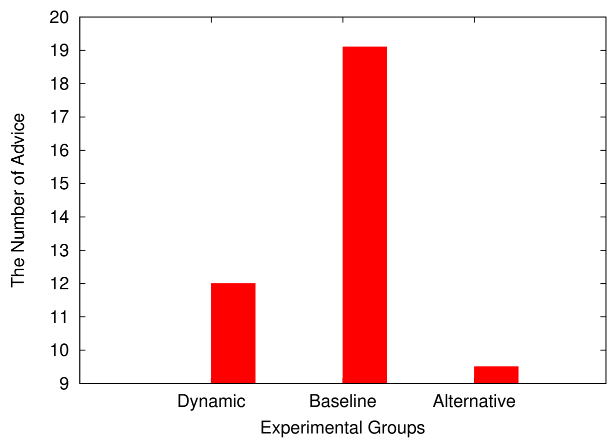
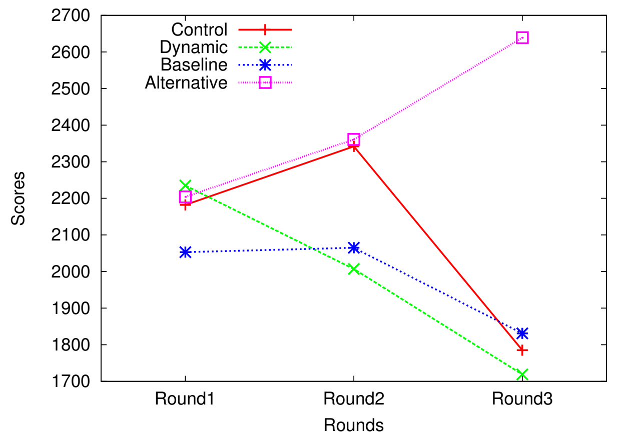


Figure 3: A trained Pac-Man agent teaches student agents. Left: Sarsa students and Sarsa teachers. Right: average advice that human players receive in Round 2. Note that Control Group doesn't provide advice during the Round 2; the number of advice for Control Group is zero.



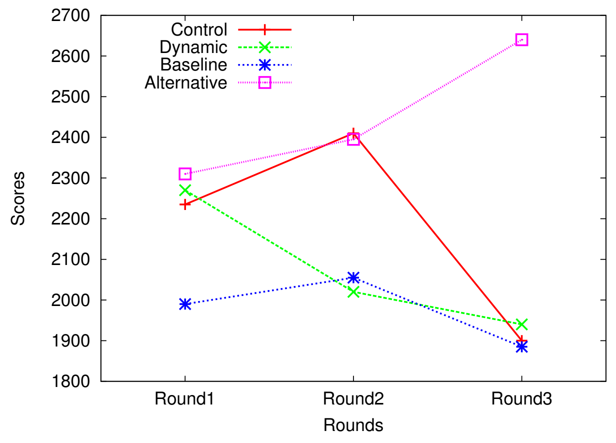
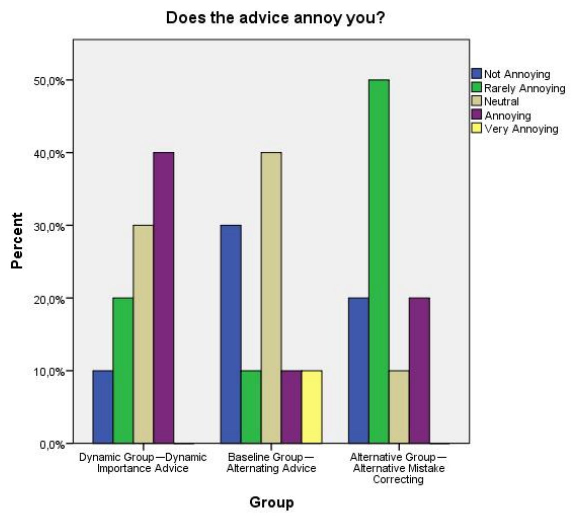


Figure 4: Left: average score among human players for each group. Right: Median score among human players for each group.



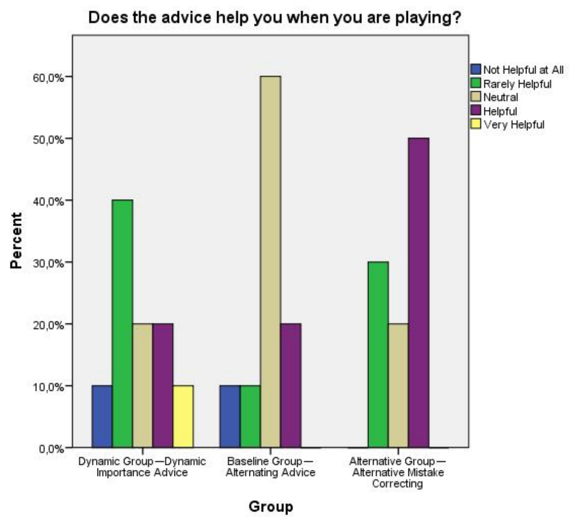


Figure 5: The survey results about the user experience and advice helpfulness. Left:The advice helpfulness survey from users. Because the Control Group doesn't have advice, these survey questions are not applicable. Right: The user experience for the advice during the experiments.

lower scores in both Figure 4 left and right, compared with Round 1 and Round 2. However, why does the Alternative Group perform better than other two groups with advice? Moreover, the Dynamic Group and the Baseline Group with advice also perform worse than the Control Group in Round 2. Why do these two groups with advice perform badly in Round 2? To answer these two questions, we need to analyze other data.

Figure 3 (right) shows the average amount of advice that the agent teacher provided to human players in Round 2. The average amount of advice for the Dynamic Group and the Baseline Group are 12 and 19, respectively. However, the each human player in the Alternative Group receives 7 advice in Round 2. Based on the fact there is no advice for Control Group, we conclude that Dynamic Group and Baseline Group excessively give suggestions to human players which tampers the user experience and annoy players, resulting in poor performance. We can also justify this argument by a survey answered by the players after the experiment (see Figure 5, left). The players in the Dynamic Group and the Baseline Group feel more annoyed than those in the Alternative Group by the advising process. 50% of the players in the Alternative Group feels the advice is rarely annoying. In contrast, 40% players in Dynamic Group and 40% players in Baseline Group think the advice is annoying and neutral, respectively.

Furthermore, human players in the Alternative Group experience less advice (and interruption to gameplay) and are willing to consider the advice from the agent teacher during the experiments. In Figure 5 (right), most of players in the Alternative Group think that the advice is more helpful than the other two groups. 50% of the players in the Alternative Group feels the advice is helpful during the experiments. However, 40% of the players in the Dynamic Group and 60% of the players in the Baseline Group had rarely helpful and neutral attitudes toward advice from the agent teacher, respectively. This explains why players in the Alternative Group perform well in Round 2 and Round 3 — they learned something useful from the agent teacher with proper amount of advice. An Eta correlation statistic was performed on the questionnaire items to examine the interaction between group membership and questionnaire responses. The largest Eta score, $Eta = .345$ was found for the question concerning the **necessity** of the advice. This shows that the perception that each player had for the necessity of the given advice was the mostly influenced attribute by the advising method they experienced (their group membership).

In all, we conclude that the advice substantially helps human players if the agent teacher gives a proper amount of advice to the human players. Otherwise, it will hurt the performance of human players.

5. DISCUSSION

In this paper, we developed new algorithms based-on previous work [10]. The experimental results lead us to the following conclusions about teaching with an advice budget.

The new algorithms are capable of teaching both agents and humans, Some of new algorithms perform better than old ones under same budget, Redistributing advice in the long run can improve the performance of agents, The amount of advice given is a key issue when human players are playing the games. The advice substantially helps human players if the agent teacher gives the proper amount of advice to the human players.

When teaching humans, the quality of the advice themselves or the advice distribution policy are not the only crucial factors for the advising performance. Other perceptional aspects such as the annoyance of the player from the given advice, also play an important role. Ideas for future work include testing new adaptive threshold advice methods for teaching humans and that will take into consideration other human factors too, and the development of new state importance metrics. In Figure 3 (left) there is a weird jump around 160 episode. We will investigate in the future work. Also, we add more information for human players such that a better path to get higher scores.

6. ACKNOWLEDGEMENTS

Thanks for the comments from anonymous reviewers and this work was supported in part by NSF IIS-1149917.

7. REFERENCES

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[2] N. Carboni and M. Taylor. Preliminary results for 1 vs. 1 tactics in starcraft. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS-13)*, May 2013.

[3] D. Chakraborty and S. Sen. Teaching new teammates. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 691–693. ACM, 2006.

[4] J. A. Clouse. *On Integrating Apprentice Learning and Reinforcement Learning*. PhD thesis, 1996. AAI9709584.

[5] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[6] P. Rohlfshagen and S. M. Lucas. Ms pac-man versus ghost team cec 2011 competition. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 70–77. IEEE, 2011.

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

[8] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

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

[10] L. Torrey and M. Taylor. Teaching on a budget: agents advising agents in reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1053–1060. International Foundation for Autonomous Agents and Multiagent Systems, 2013.