Connection Science
Vol. 00, No. 00, Month 200x, 1–20

RESEARCH ARTICLE

Reinforcement Learning Agents Providing Advice in Complex Video Games

Matthew E. Taylor[a*] , Nicholas Carboni[b], Anestis Fachantidis[c], Ioannis Vlahavas[c], and Lisa Torrey[d]

[a] *Washington State University, Pullman, WA, USA*;
[b] *PHD Virtual Technologies, Morris Plains, NJ, USA*;
[c] *Aristotle University of Thessaloniki, Thessaloniki, Greece*;
[d] *St. Lawrence University, Canton, NY, USA*

(*Received 00 Month 200x; final version received 00 Month 200x*)

This article introduces a teacher-student framework for reinforcement learning, synthesizing and extending material that appeared in conference proceedings [22] and in a non-archival workshop paper [6]. In this framework, a teacher agent instructs a student agent by suggesting actions the student should take as it learns. However, the teacher may only give such advice a limited number of times. We present several novel algorithms that teachers can use to budget their advice effectively, and we evaluate them in two complex video games: StarCraft and Pac-Man. Our results show that the same amount of advice, given at different moments, can have different effects on student learning, and that teachers can significantly affect student learning even when students use different learning methods and state representations.

Keywords: Reinforcement Learning, Agent Teaching, Pac-Man, StarCraft

1. Introduction

Using reinforcement learning (RL), agents can autonomously learn to master sequential-decision tasks. In these tasks, an agent must develop a control policy for taking actions in an environment. RL agents have traditionally been trained and used in isolation, but research is beginning to produce ways for them to interact productively with other agents and with humans.

This work focuses on how an RL agent could serve as a teacher for a task it has mastered. We begin with another RL agent in the role of the student, but we prefer teaching approaches that could potentially be used with human students as well. This limits us to human-understandable teaching methods, prevents teachers from assuming any familiarity with a student's internal workings, and prevents students from simply starting with the teacher's knowledge. Furthermore, it requires teachers to be able to instruct students that may learn and perceive their environment differently.

As a motivating example for RL agents as teachers, consider the fast-growing industry of computer games. One measure of a successful game is how many humans choose to learn to play it. Modern games often have built-in training sessions to help them; currently, this is additional content created by game developers. Instead, perhaps RL agents could learn to play these games autonomously and then teach

*Corresponding author. Email: taylorm@eecs.wsu.edu

ISSN: 0954-0091 print/ISSN 1360-0494 online

DOI: 10.1080/09540090xxxxxxxxxxxx
http://www.informaworld.com

human players. This could reduce the amount of developer time required to produce training content.

There are many possible ways to help agents learn [3, 20], 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. We advocate this method because it requires minimal similarity between teachers and students — only a common action set. They may use different learning algorithms, and they may have different ways of representing the state of their environment. This is important for the long-term goal of having human students, but it is also important to enable agents with different implementations (e.g., created by different companies) to teach each other without significant re-engineering.

Another assumption we make for this type of teaching is that teachers cannot give unlimited quantities of advice. One reason for this restriction is that human students would have limited patience and attention. However, it is also true that some domains limit communication between agents. Furthermore, a teacher that over-advises a student could actually hinder its learning, if the differences between them are significant.

This article studies how an RL agent can best teach another RL agent using a limited amount of advice. The teacher observes the student and can give advice a fixed number of times, but cannot observe or change anything internal to the student. We propose a set of teaching algorithms: *early advising*, *alternate advising*, *importance advising*, *mistake correcting*, and *predictive advising*. We evaluate these algorithms experimentally in two domains: StarCraft and Pac-Man. The results show that the same amount of advice, given at different moments, can have different effects on student learning, and that teachers can significantly affect student learning even when students use different learning methods and state representations.

This article extends work in [6] and [22] by proposing two new measures for importance advising, variance-based importance, and the absolute deviation importance, applying them to the Pac-Man domain. Moreover, this work applies standard importance advising, its two new measures, and the mistake correcting algorithm in the complex domain of Starcraft.

2. 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. Discrete RL algorithms make updates to individual Q-value entries in this table. However, in some larger environments, the states cannot be enumerated. In such cases, the states (or the state-action pairs) are 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, Sarsa(λ), and Q(λ) with linear Q-function approximation. These are well-known RL algorithms that can incorporate advice with minimal modification. Since they already allow for exploratory actions, they 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 parameter values in each task for reproduceability, but we direct readers elsewhere for a detailed discussion of the algorithms [18].

The weights $\{w_1, w_2, ...\}$ 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 habit and can benefit much more from advice. The experiments in this article therefore use pessimistic initialization.

3. Teaching on a Budget

Suppose that an RL agent has learned an effective policy π for a task. Using this fixed policy, it will teach another RL agent that is beginning to learn the same task. As the student learns, the teacher will observe each state s the student encounters and each action a the student takes. The teacher may perceive the states differently than the student does. In n of the states, the teacher may advise the student to take what it sees as the correct action: $\pi(s)$.

How should the teacher spend its advice most effectively? Calculating an optimal strategy is unlikely to be feasible beyond the simplest of RL problems. We instead take an experimental approach to this question, proposing and testing several heuristic algorithms for deciding when to give advice.

3.1. *Early Advising*

It seems clear that students should benefit more from advice early on, when they know very little. Applying this intuition, our first approach has the teacher give advice in each of the first n states the student encounters. This approach, which we call *early advising*, is Algorithm **1**. It serves as our baseline.

3.2. *Alternating Advice*

With early advising, the advice budget is spent quickly in a limited part of the state space. To address these potential shortcomings, our second approach has the teacher give advice once every m steps for the first nm states the student encounters. This approach, which we call *alternate advising*, is Algorithm **2**. It allows the student to explore more of the state space in the vicinity of the teacher's policy before the advice budget is exhausted.

Algorithm 1 Early Advising

```
1: procedure EarlyAdvising(π, n)
2:     for each student state s do
3:         if n > 0 then
4:             n ← n − 1
5:             Advise π(s)
6:         end if
7:     end for
8: end procedure
```

Algorithm 2 Alternate Advising

```
1: procedure AlternateAdvising(π, n, m)
2:     step ← 0
3:     for each student state s do
4:         if n > 0 and step mod m = 0 then
5:             n ← n − 1
6:             Advise π(s)
7:         end if
8:         step ← step + 1
9:     end for
10: end procedure
```

Algorithm 3 Importance Advising

```
1: procedure ImportanceAdvising(π, n, t)
2:     for each student state s do
3:         if n > 0 and I(s) ≥ t then
4:             n ← n − 1
5:             Advise π(s)
6:         end if
7:     end for
8: end procedure
```

3.3. *Importance Advising*

When all states in a task are equally important, 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. Consider that games often have calmer and tenser moments. In certain situations, the right move can win the game or the wrong move can lose it; in others, any move is acceptable and none are disastrous. This is an intuitive definition of state importance, which we will soon quantify with a function $I(s)$.

A teacher that is conscious of state importance could give advice only when it reaches some threshold t. We call this approach in Algorithm **3** *importance advising*.

Because our teachers are RL agents with Q-functions, they have a natural way to calculate $I(s)$. Recall that a Q-value $Q(s, a)$ is an estimate of the rewards ultimately achievable by taking action a in state s. If the Q-values for all the actions in s are the same, then it does not matter which one is taken, and s is unimportant. However, if some actions in s have larger Q-values than others, then it does matter, and s has some importance. We therefore propose the following definition of state importance:

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

This measure was first introduced by Clouse in his work on apprenticeship learning [8], but it was used there to approximate a learner's confidence in a state. Here, we compute $I(s)$ with the teacher's fully-learned Q-function rather than the student's partially-learned one, and in this context it is a better indicator of state importance than agent confidence.

Moreover, in this text we propose two additional novel importance measure to respond to the limitations presented by the first measure. The novel measures consider range of the Q-values for a given state — the range of Q-values is a biased estimator of their dispersion and is lossy because it does not provide any information on the actual distribution. For instance, the Q-values could be uniformly distributed, or all of them but one may be very close to their maximum, and the difference between the highest and lowest Q-values could be the same in both cases.

The first novel importance advising measure proposed here uses the variance statistic instead of the range to measure the importance of a state. The variance (σ^2) is a natural formulation of the inherit risk of a decision and is considered a more robust and less biased statistic than the range. This version of the algorithm computes the variance of the teacher's Q-values in the current state and compares it to a threshold. Variance-based importance is defined as:

$$I_V(s) = \frac{1}{|A|} \sum_{i=1}^{|A|} (Q(s,a) - \overline{Q(s,a)})^2$$

where $|A|$ is the number of actions in the action set A and $\overline{Q(s,a)}$ is the mean of the Q-values for the given state s.

The second novel importance measure is based on the absolute deviation of the Q-values, which is considered more suitable than the standard deviation when normality guarantees do not hold. The absolute deviation importance measure addresses two potential problems of the variance-based importance measure. First, because the deviations of Q-values from the mean Q-Value are squared, this gives more weight to extreme values (i.e., more importance to states with some extreme action values). Second, tuning the threshold for a squared value is more difficult and sensitive to small changes. The absolute deviation-based importance measure is defined as:

$$I_D(s) = \frac{1}{|A|} \sum_{i=1}^{|A|} |Q(s,a) - \overline{Q(s,a)}|$$

3.4. *Mistake Correcting*

Even if a teacher saves its advice for important states, it may end up wasting some advice in states where the student had already intended to take the correct action. Advice can only have an effect when used in states where the student would otherwise have made a mistake. If teachers could restrict their advice to these states, they should be able to improve upon the above methods.

However, one of our key assumptions in this work is that teachers have no direct access to student knowledge. To make it possible for teachers to spend advice exclusively on mistakes, students would need to announce their intended actions

Algorithm 4 Mistake Correcting

```
1: procedure MistakeCorrecting(π, n, t)
2:     for each student state s do
3:         Observe student's announced action a
4:         if n > 0 and I(s) ≥ t and a ≠ π(s) then
5:             n ← n − 1
6:             Advise π(s)
7:         end if
8:     end for
9: end procedure
```

Algorithm 5 Predictive Advising

```
1: procedure PredictiveAdvising(π, n, t)
2:     for each student state s do
3:         Predict student's intended action a
4:         if n > 0 and I(s) ≥ t and a ≠ π(s) then
5:             n ← n − 1
6:             Advise π(s)
7:         end if
8:     end for
9: end procedure
```

in advance and give teachers an opportunity to correct them. This introduces additional communication into the framework that may not be convenient in all situations, but the approach serves as a useful upper bound. We call this approach, in Algorithm **4**, *mistake correcting*.

3.5. *Predictive Advising*

Although teachers cannot directly access student knowledge, they may be able to infer student policies from their behavior. A teacher observes the states a student encounters and the actions it takes. Using these observations as training data, the teacher can train a classifier to predict student actions, and use these predictions in place of student announcements. We call this approach, in Algorithm **5**, *predictive advising*.

This approach approximates mistake correcting, but has the advantage of not requiring additional communication from the student. If a teacher's action predictor performs perfectly, predictive advising becomes equivalent to mistake correcting. When it makes inaccurate predictions, the teacher sometimes wastes advice, making predictive advising more like importance advising. Inaccurate predictions can also make the teacher miss opportunities to give useful advice, which importance advising would not have missed.

Many algorithms for supervised learning could potentially be applied to this classification task. In this article we use a Support Vector Machine, as implemented in the *SVM-Light* software package [9]. Action prediction can be addressed as a partial ranking problem: given the student's state (as the teacher sees it), the teacher attempts to predict which action the student will rank more highly than the rest. SVM-Light's ranking SVM is well-suited to this problem.

Each observed state-action pair generates one training example. Suppose a teacher observes a state s with k features, and sees the student choose one of 3 possible actions. The corresponding training example generated for the ranking

SVM would be structured as:

2 $f_1(s, a_1)$, $f_2(s, a_1)$, ..., $f_k(s, a_1)$
1 $f_1(s, a_2)$, $f_2(s, a_2$, ..., $f_k(s, a_2)$
1 $f_1(s, a_3)$, $f_2(s, a_3)$, ..., $f_k(s, a_3)$

In this example, each action is represented by one line of features. The numbers in the left column specify pairwise ranking constraints between actions. Because action a_1 is higher than a_2 and a_3, the SVM applies the constraints $Q(s, a_1) > Q(s, a_2)$ and $Q(s, a_1) > Q(s, a_3)$. Because actions a_2 and a_3 have the same values, no constraints are generated between them. This reflects the teacher's knowledge, which is only that the student chose a_1 over the other actions.

The output of the ranking SVM, when queried on a state s, is a set of real numbers, one for each action available in s. The predicted student action in s is the one with the highest number. We keep most of the SVM-Light parameters at their default values, except for the margin/error tradeoff C, which we experimentally tuned to 1000.

The teacher trains a new SVM after each episode, using training examples from the previous episode. Average SVM training times are approximately one second. This is an inconspicuous delay between episodes, but it could be disruptive during episodes, which is why we do not update the SVMs more often.

Note that this classification task is inherently challenging for several reasons. First, students are constantly learning and will likely produce inconsistent training data because their behavior is non-stationary. Second, students sometimes take random exploration steps, which means the data will be noisy. Third, the student's state representation can be different from the teacher's, which means the hypothesis space of the classifier may not even contain the student's policy. Despite these challenges, our results will show that useful predictions are achievable in some scenarios.

4. Experimental Domains

We evaluate these teaching algorithms in two complex, stochastic experimental domains. Due to space limitations, we cannot fully describe all the details of our implementations. However, all of our code is available at the corresponding author's website.[1]

4.1. *StarCraft*

StarCraft is a popular real-time strategy game that simulates a war between two armies. Successfully playing it involves juggling many (sometimes conflicting) priorities, including resource collection, building and unit construction, and technology upgrades. This work focuses on a more manageable subgame involving one-on-one combat between two units. Figure 1 shows the center of the board used in experiments. This map consists of one island on which the two units fight. There is also a barrier through which they cannot pass.

A Terran Marine and a Zerg Zergling begin the episode at fixed start locations. The primary difference between these units is that the Marine is a ranged unit and the Zergling is not; the Zergling must be within close range of its enemy to attack. The Marine is the learning agent, while the Zergling is controlled by the standard game AI: it is stationary until the Marine is close enough to be seen, or until the

[1] `http://eecs.wsu.edu/~taylorm/13ConnectionScience.html`

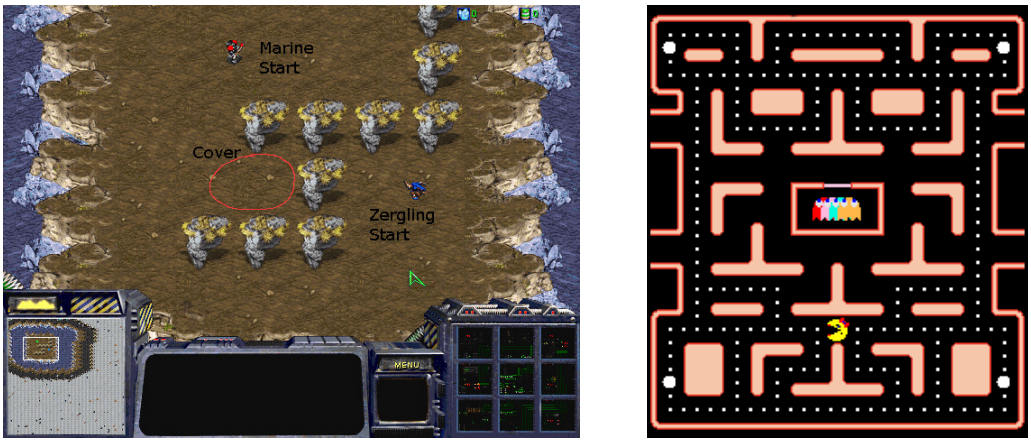

Figure 1. The screenshot on the left shows the 1-vs.-1 StarCraft map and highlights the area from which the Marine can harm the Zergling without immediate retaliation. The figure on the right shows the Pac-Man maze.

Marine shoots at it. The episode ends when one of the units dies, or a maximum of 1000 actions have been executed by the agent.

If the Marine could simply run around the barrier and kill the Zergling immediately, then our problem would be trivial. However, if the Marine were to get too close he would surely die; hit point and damage values are set so that in a close-range fight the Zergling will always win. The ideal policy for the Marine is therefore to attack the Zergling from over the barrier and kill it before it is able to reach him.

The agent represents the current state using six state variables:

(1) the agent's X position,
(2) the agent's Y position,
(3) the straight line distance to the enemy,
(4) the difference in hit points between the agent and the enemy,
(5) a boolean value for whether the enemy is stationary, and
(6) and the angle of the enemy relative to the agent.

The agent receives a reward of -0.3 on every step.[1] At the end of an episode (when the agent does not exceed 1000 actions), we calculate the difference in the health of the agent and the enemy as the reward for the final step of the episode. Episodes often last the full 1000 actions during initial learning because the agent never engages the enemy; in this case the agent accumulates a total reward of -300 due to the step penalty. Some agents converge to a policy which suggests walking directly towards the enemy to kill themselves and lose the episode as quickly as possible, avoiding the penalty for taking a large number of steps. This strategy results in a total reward of roughly -15. In the best-case scenario, the agent will kill the enemy from behind cover without being hit, and receive a total reward of roughly 20.

In any state the agent can execute one of seven actions. For one time step the agent can:

(1) take no action,
(2) attack the enemy,
(3) move towards the enemy,
(4) move south,

[1]Even though there is a penalty for every step, there are cases where taking no action can be advantageous (e.g., when the enemy is moving).

(5) move north,
(6) move east, or
(7) move west.

The attack command will cause the Marine to shoot the enemy if it is within range, or move towards the enemy if is not. Because the Zergling will quickly kill the Marine if the Marine does not begin its attack from cover, the attack action and the move towards enemy action often result in the agent's death during early experimentation, making it particularly difficult for the agent to learn to correctly attack the enemy.

StarCraft, like most strategy games, has a "fog of war" setting that deliberately obscures units and structures from the players' views. In our experiments, we disabled this setting. If the fog of war were enabled, and the Marine had not yet moved close enough to the Zergling, the actions for attacking the enemy and moving towards the enemy would have no effect.

The agent learns using Sarsa with a fixed exploration rate of 0.05 and a fixed learning rate of 0.15, tuned by running multiple trials with approximately 10 different parameter settings. A CMAC [2] tile coding is used for function approximation, where each of the 6 state variables are tiled independently with 32 tilings.

4.2. *Pac-Man*

Pac-Man is a 1980s arcade game in which the player navigates a maze like the one in Figure 1 (right), trying to earn points by touching edible items and trying to avoid being caught by the four ghosts. We use an implementation of the game provided by the Ms. Pac-Man vs. Ghosts League [16], which conducts annual competitions. Ghosts in this implementation chase the player 80% of the time and move randomly the other 20%.

Pac-Man episodes all occur in the same maze. The agent has four actions — move up, down, left, and right — but in most states only some of these actions are available. The agent can (roughly) maintain its position by toggling quickly between two opposite actions (i.e., selecting the move right action and then immediately selecting the move left action). 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, and also cause the ghosts to 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 causes the ghost to respawn in the lair at the center of the maze. The episode ends if any ghost catches Pac-Man, or after a maximum of 2000 steps.

This domain is discrete but has a very large state space. There are 1293 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.

Useful high-level features tend to describe distances between Pac-Man and other objects of interest. Action-specific features are more useful than global features. For example, a global feature might be "the distance from Pac-Man to the nearest food pellet." Making this feature specific to action a, it becomes "the distance from Pac-Man to the nearest food pellet after Pac-Man executes a."

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 also to normalize the features to the range $[0, 1]$.

We create agents with different state representations in this domain by defining two distinct feature sets. One feature set consists of 16 features that count objects at a range of distances from Pac-Man. The other consists of 7 heavily-engineered distance-related features. These features are not fully documented here for space reasons, but their implementation is available in the on-line appendix.

A perfect score in an episode would be 5600 points, but this is quite difficult to achieve (for both humans and agents). An agent executing random actions earns an average of 250 points. The 16-feature set described above allows an agent to reach an average of 2600 points per episode, successfully eating most of the pellets and the occasional edible ghost. The 7-feature set allows an agent to learn to catch more edible ghosts and achieve a per-episode average of 3800 points. We therefore refer to the 16-feature set as "low-asymptote" and the 7-feature set as "high-asymptote."

5. Teaching Results

This section demonstrates improvements in student learning via teaching in the StarCraft and Pac-Man tasks. First, we train agents independently in these tasks, and select the best-performing agents to be teachers. Then, we have students learn the tasks, with advice from the teachers. The learning curves for the students are what we show here.

To smooth the natural variance in student performance, each learning curve is an average over multiple independent trials of student learning. In Pac-Man, each curve is an average of 30 independent trials. In StarCraft (for which simulation is much slower) there are 15 independent trials, but the points on the learning curves are further smoothed by averaging over a 5-episode moving window.

While training, an agent periodically halts its training to run evaluation episodes, in which no learning, exploration, or teaching occurs. In StarCraft, every other episode is an evaluation episode. In Pac-Man, we run 30 evaluation episodes after every 10 learning episodes. In both cases, the learning curves only reflect student knowledge, not teacher knowledge.

We consider one learning method to be better than another if its learning curve has a steeper slope or a higher asymptote. Agents that use different learning algorithms, state representations, or parameter settings can differ in both of these ways. Our experiments focus on the impact of teaching algorithms, keeping other learning and experimental parameters fixed. The parameters and thresholds of the learning algorithms in the next sections were tuned through preliminary experimentation.

To measure learning speed in a holistic way, we compute areas under learning curves. The total rewards are compared using t-tests on their areas with $\alpha = 0.05$. When we report that the difference between two curves is significant, it means there is at least 95% confidence that one curve has higher total reward.

5.1. *StarCraft*

This section examines teaching in the StarCraft task with only 50 pieces of action advice. To evaluate the effects of teacher quality, we do so with two different teachers. One is a "good" teacher, trained for 500 episodes to achieve an average reward of 16.2, with a standard deviation of 6.7. The other is a "poor" teacher, trained for only 200 episodes, achieving an average reward of -10.7, with a standard deviation of 13.

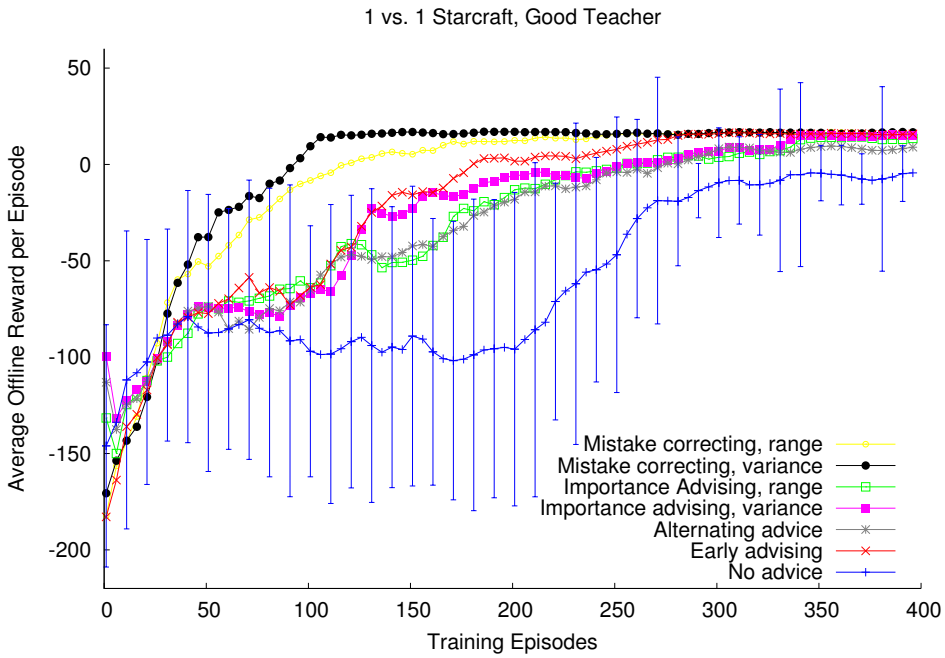

Figure 2. StarCraft learning with a good teacher.

5.1.1. Using a Good Teacher

Figure 2 shows how a good teacher can impact the learning of a StarCraft student. The *Average Teacher Performance* line shows the average performance of the teacher. The *No Advice* curve shows the average performance of independent Sarsa agents without teaching. The other curves show that early advising, alternate advising, importance advising, and mistake correcting all outperform independent learning. Typically, all of these algorithms lead the Marine to consistently kill the Zergling from behind cover before it was wounded.

Standard error bars are shown every 10 episodes on the *No Advice* curve to give an idea of the variation in student performance. In the other learning curves, the error bars are similar, and are omitted for readability. Early advising produces a statistically significant improvement over independent learning. This is also true for alternate advising and importance advising (with a threshold of $t = 24$), but there is no significant difference between these algorithms and early advising.

For variance-based importance advising, the variance threshold was set to $t = 75$ and the results showed a statistically significant improvement over No Advice both in the total reward and the reward after 400 training episodes. However, no statistically significant difference was observed compared to the above methods teaching methods.

Finally, consider the Mistake Correcting algorithm, using both the range and the variance statistic. Mistake correcting provides advice only when the student's intended action is different from the teacher's action and the state is considered important. For the range statistic, the threshold was set to $t = 15$ and for variance, $t = 45$. Both versions of mistake correcting showed a statistically significant performance improvement ($p < 0.05$) compared to all the other methods presented in this experiment, and also compared to No Advice. Although the variance method shows better performance than the range method, the difference is not statistically significant at $p < 0.05$

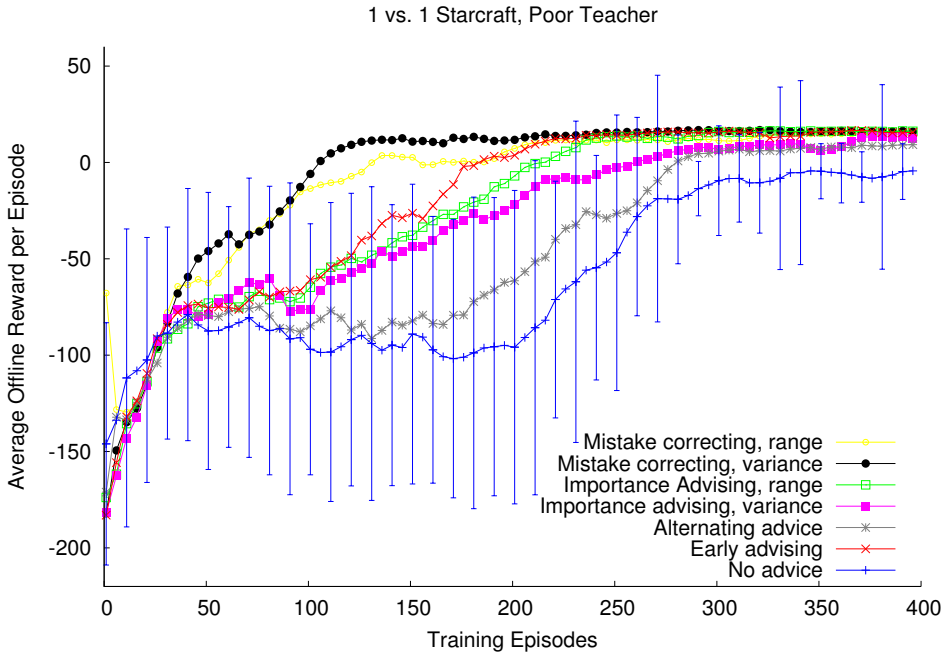

Figure 3. StarCraft learning with a poor teacher.

5.1.2. Using a Poor Teacher

Figure 3 shows how a poor teacher can impact the learning of a StarCraft student. Because this teacher's performance is significantly worse than the teacher in the previous section, we expected student performance to significantly decrease. However, we found the relative ordering of the algorithms to be very similar. At the 95% confidence level the same statistically significant differences were found between the algorithms as those of using a good teacher. However, no statistically significant difference were found between the early advising, the standard importance advising (i.e., range importance measure) and the variance-based importance advising.

When the students follow the teacher's advice, they achieve a reward very similar to that of the poor teacher. However, when the agents execute their learned policy (as graphed in Figure 3), the agents eventually reach a higher performance than the teacher because they have learned to quickly kill themselves. This local maximum is better than continually wandering around the state space until the episode ends, but is much worse than correctly killing the enemy. Because the agent stumbles upon this sub-optimal local maximum, it actually makes learning the optimal policy much harder.

It is important to note the significantly worse performance of alternate advising using a poor teacher. One intuition is that a poor teacher (i.e., a teacher with Q-values that have not converged) increases the importance of having an advising criteria such as those of importance advising and mistake correcting. However, early advising does still perform well. This behavior exposes the fact that a teacher's ability may affect which teaching method is most effective.

5.2. *Pac-Man*

In this section, we test teaching in the game of Pac-Man. Because experiments run significantly faster than in StarCraft, we can run experiments for many more episodes and be assured asymptotic performance is reached. Thus, we do not expect

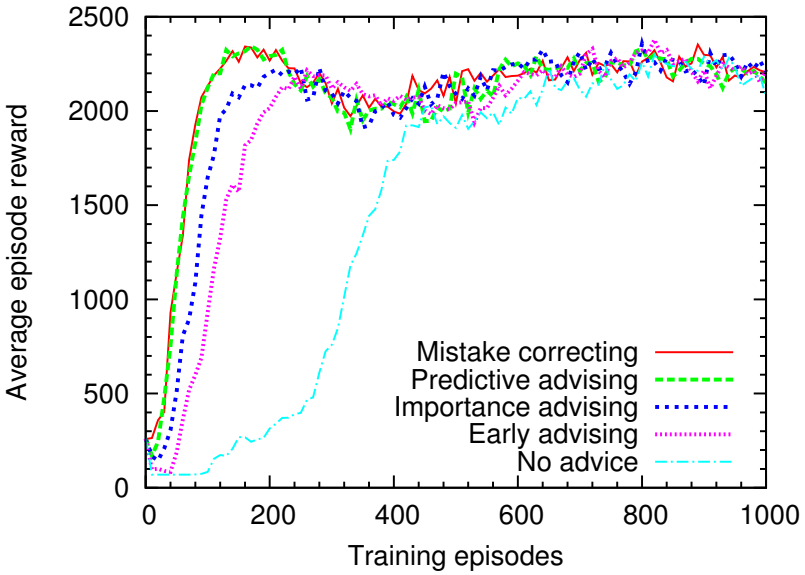

Figure 4. Pac-Man learning with similar students and teachers using Sarsa with the low-asymptote feature set.

teaching to change the asymptotic performance of students, but we do look for it to improve their learning speed.

With the exception of early advising, all of our teaching approaches have a parameter t, the threshold above which a state is considered important. To explore how t affects performance, we try 10 values for each teacher, uniformly distributed across that teacher's $I(s)$ range. For each teacher-student pair, we report the most effective value for t.

Pac-Man teachers are given an advice budget of $n = 1000$, which is roughly half the number of steps in a single well-played episode. Since independent learners take between 400 and 800 training episodes to learn this game, Pac-Man students are only receiving advice during a small fraction of their training steps. The RL parameters that all Pac-Man agents use are $\epsilon = 0.05$, $\alpha = 0.001$, $\gamma = 0.999$, and $\lambda = 0.9$.

First, we present experiments where the teacher and student use the same algorithm and feature set. In Figure 4, both agents use Sarsa(λ) and the low-asymptote 16-feature set; $t = 50$. Differences between curves are significant for all pairs except mistake correcting and predictive advising.[1] In Figure 5, both agents use Sarsa(λ) and the high-asymptote 7-feature set; $t = 200$. Differences between curves are significant for all pairs except early advising and independent learning.

Although these teachers are giving advice in only a small fraction of the training steps, some of them have significant effects on student learning. Advice has a higher overall impact on students with the 16-feature set because these students have a simpler policy to learn. Early advising provides a large benefit with the 16-feature students, but not with the 7-feature ones. Importance advising is slightly but consistently better than early advising, and the best t thresholds are above 0, which confirms that saving advice for important states can be effective. Mistake correcting consistently outperforms importance advising, which confirms that saving advice for mistakes is also effective. Predictive advising also outperforms importance ad-

[1]Predictive advising was easier to implement in Pac-Man than in StarCraft. This is because the Pac-Man simulation runs in Linux, rather than Windows, and there are many fewer actions in Pac-Man, making prediction easier. Implementing prediction in StarCraft is left to future work, although we expect results in the Pac-Man domain.

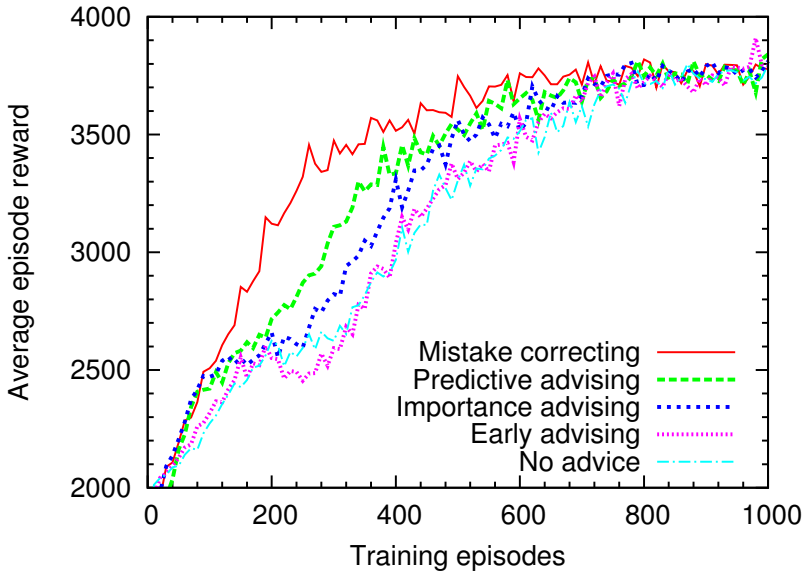

Figure 5. Pac-Man learning with similar students and teachers using Sarsa with the high-asymptote feature set.

vising, and for the 16-feature students it even matches mistake correcting. These results suggest that mistake correcting is the best choice if it is feasible, and if not, predictive advising is the best alternative.

The good performance of predictive advising in Figure 4 is not due to perfect action prediction. In fact, prediction accuracy is lower (79%) than in Figure 5 (86%). But prediction errors are less costly when teaching low-asymptote students because they benefit more from any advice schedule. For the same student, increased prediction accuracy corresponds to better performance with predictive advising. However, the impacts of wasted and delayed advice are not the same for all students.

5.2.1. Increased Differences Between Student and Teacher

Our next experiments investigate the effects of having the teacher and student use different learning algorithms. This factor would be irrelevant if all algorithms converged to the same optimal policy, but in practice this is not the case. Q(λ) and Sarsa(λ) produce asymptotic policies for Pac-Man whose performances differ by approximately 200 points. They also learn at different speeds; for the sake of variety, we exaggerate this difference by using $\lambda = 0.7$ in Q(λ) to slow it further.

In Figure 6, a Q(λ) teacher advises a Sarsa(λ) student; $t = 20$ and prediction accuracy is 80%. Differences between curves are significant for all pairs except early advising and importance advising. Figure 7 shows the performance when the algorithms are reversed; $t = 100$, and prediction accuracy is 81%. Differences between curves are significant for all pairs except mistake correcting and predictive advising. All of these agents use the 16-feature set.

Although these students use different algorithms and progress at different rates than their teachers, the differences do not appear to hinder teaching. Advice has a higher overall impact on students that use Sarsa(λ), probably because advice is treated as exploration, and Sarsa(λ) takes exploration into account more than Q(λ) does. These results suggest that the learning algorithm of the student has more impact on the effectiveness of teaching than the learning algorithm of the teacher does. While some students may respond better to advice than others, teachers can effectively advise students that learn differently.

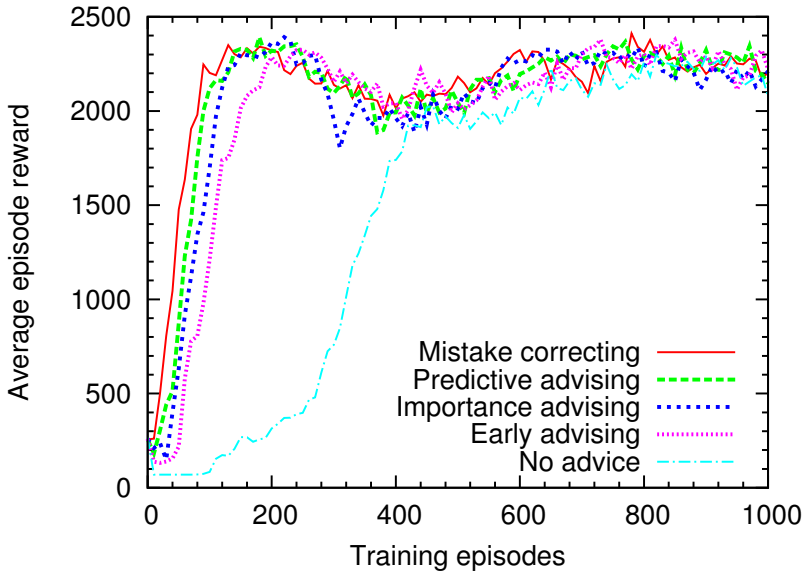

Figure 6. A Q-learning teacher advises a Sarsa student, both using the low-asymptote feature set.

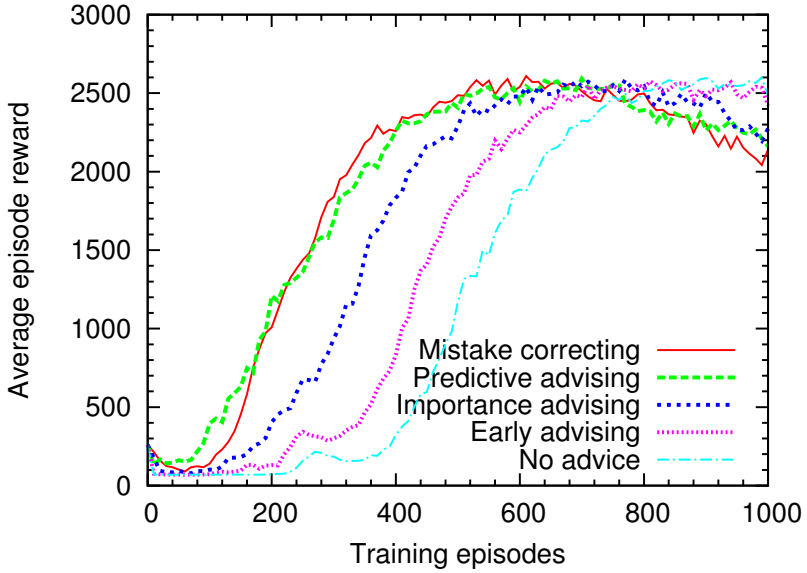

Figure 7. A Sarsa teacher advises a Q-learning student, both using the low-asymptote feature set.

Finally, we present experiments where the teacher and student use different feature sets. We expect this to be the most challenging type of difference because it causes large differences in the asymptotic performance of teachers and students. It is not obvious that advice will be helpful across this divide, and there is even the risk that it might be harmful.

In Figure 9, a high-asymptote 7-feature teacher advises a low-asymptote 16-feature student; $t = 100$. Differences between curves are significant for all pairs. In Figure 9, the feature sets are reversed and $t = 250$. Mistake correcting and predictive advising are significantly different from each other and the rest, but the other three approaches are statistically equivalent. All of these agents use Sarsa(λ).

Although these teachers perceive their environment differently than their students, some of them still provide significant benefits on student learning. These effects are not always as strong as when teachers and students used the same

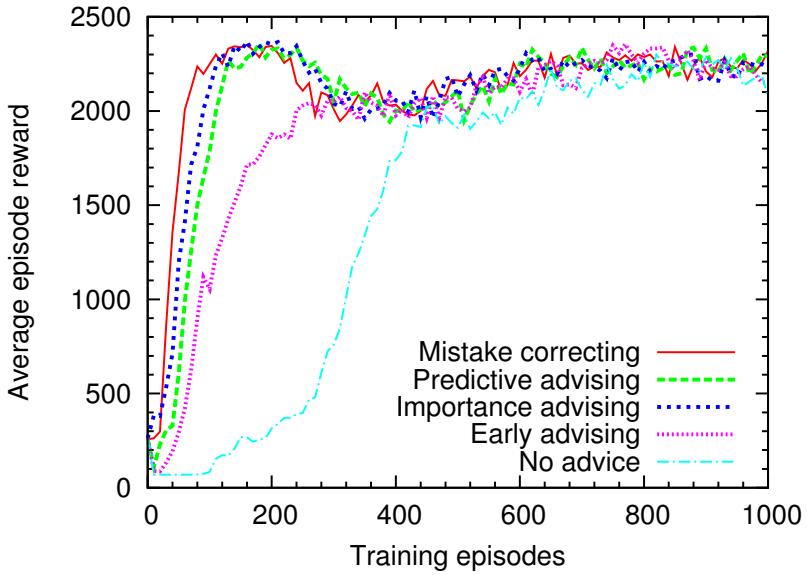

Figure 8. A teacher with a high-asymptote representation advises a low-asymptote student, both using Sarsa.

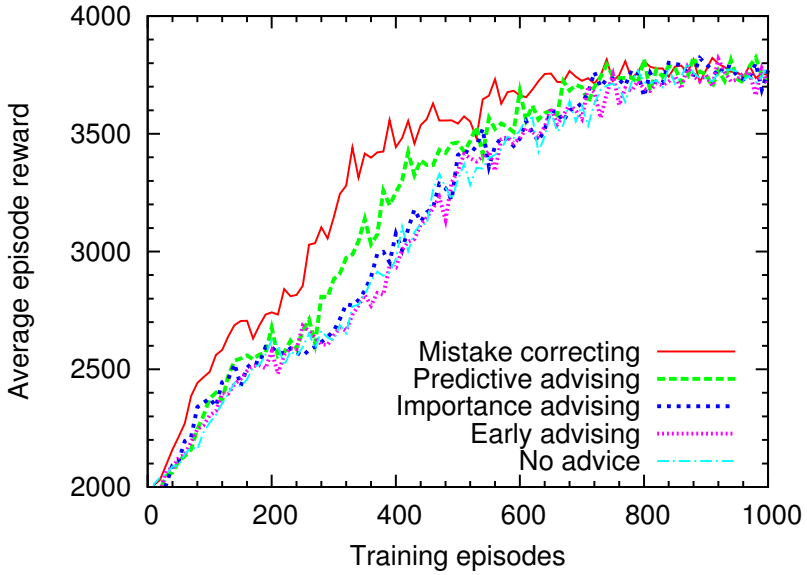

Figure 9. A teacher with a low-asymptote representation advises a high-asymptote student, both using Sarsa.

state representation, but some of them remain quite useful. Unsurprisingly, high-asymptote teachers have larger effects on low-asymptote students than vice versa. But low-asymptote teachers do have positive impacts on high-asymptote students (with mistake correcting and predictive advising), and these students then go on to outperform their teachers, as they should given their higher inherent capability. None of these teachers have negative impacts on students.

The high-asymptote agents have substantial difficulty predicting the actions of the low-asymptote agents (accuracy 61%). This causes predictive advising to perform slightly below importance advising in Figure 8. There is no such difficulty in the reverse scenario (i.e., Figure 9, which has an accuracy 86%). Prediction accuracy across feature sets is likely to depend on the specifics of the features. However, these results suggest that teachers can effectively advise students that perceive the world differently.

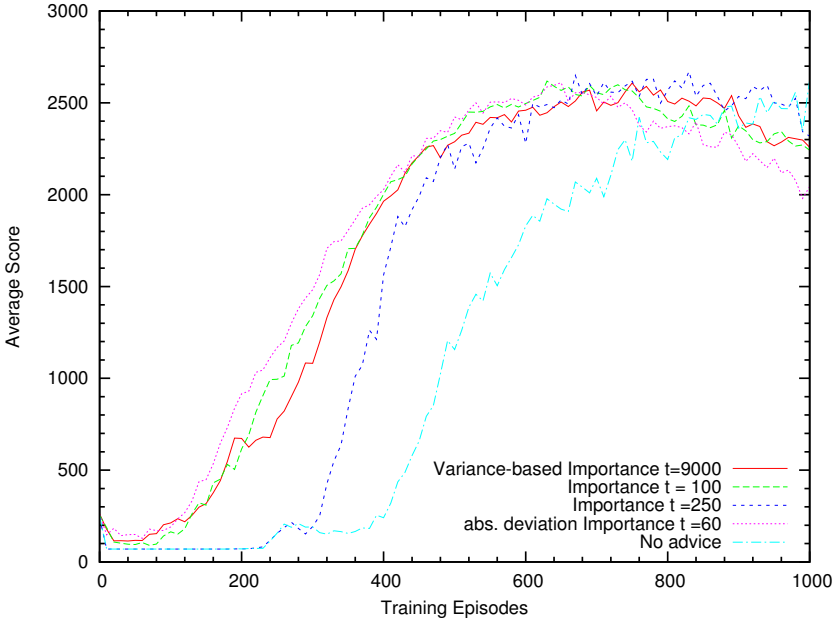

Figure 10. This figure shows results of teaching Pac-Man with different definitions of state importance.

The rate at which teachers spend their advice is partly controlled by the importance threshold t. When teaching 16-feature students, the best teachers give most of their advice within the first 10 episodes of student training, because the low-asymptote students benefit most from advice very early in their learning. When 16-feature teachers advise 7-feature students, they also do best to spend their advice quickly, before the high-asymptote students surpass them. However, when 7-feature teachers advise 7-feature students, they can perform better by giving less frequent advice over longer periods. The best settings of importance advising, mistake correcting and predictive advising spread their advice over 20, 100 and 60 episodes respectively.

5.2.2. Alternative Importance Metrics

One important open question is how to best define state importance. While this question will be fully discussed only in future work, we tested the two other importance metrics, as discussed in Section 3.3.

Figure 10 shows a Q-learning student using the low-asymptote feature set and Importance advising. Two different threshold values for the standard state importance metric are compared to the two additional importance measures. However, neither of the new metrics showed a statistically significant difference ($p > 0.05$) with the normal state importance definition (for a threshold of 100). It is important to note the poor performance of importance advising using the range statistic and a high threshold of 250. The high threshold allows less (or no) advice to be given and the performance of the student approaches that of learning without a teacher. On the other hand a very low threshold in an importance metric will allow giving most of the advice early so that the student's performance approaches that of Early Advising.

5.3. *Results Summary*

Experimental results conducted in two complex domains lead us to the following conclusions about teaching with an advice budget.

(1) Student learning can be improved by receiving small amounts of advice from a teaching agent.
(2) Advice can have greater impact when it is used in more important states, where "importance" can be defined in multiple ways.
(3) Advice can have greater impact when it is spent on correcting student mistakes.
(4) When teachers can successfully predict student mistakes, they can spend their advice budget more effectively.
(5) Teaching can improve student learning even when agents have different learning algorithms or state representations.
(6) Students can benefit from advice even from teachers with less inherent ability, and then go on to outperform their teachers.

6. Related Work

There is a growing body of work on improving reinforcement learning by leveraging knowledge from outside sources. We distinguish our work by focusing on methods that 1) are directed by an agent serving as a teacher, 2) aim to help a student maximize an environmental reward, 3) can potentially be used with human students (i.e., do not require unlimited communication, direct knowledge access, or identical state representations), and 4) and are applied to large, non-trivial tasks.

Transfer learning in RL [20] has an agent use knowledge from a source task to aid its learning in a target task. Agents performing transfer often have direct access to source-task knowledge, such as the source-task policy. In contrast, our work assumes student agents have strict limits on access to teacher knowledge.

Experience replay [12] has a student train on the recorded experiences of a teacher. This requires the student and teacher to have identical state representations, which is a limitation our methods avoid.

Apprentice learning [8] has a student ask a teacher for advice whenever its confidence in a state is low. Similarly, *advice exchange* [14] has peers ask for advice from each other based on heuristics of self-confidence and trust. Our work diverges from these by having a teacher decide when advice will occur, rather than a student.

Our own prior work involves observing a teacher performing a task repeatedly and then summarizing the behavior via rules using the student's state description. This method successfully allows the student and teacher to have different state representations (i.e., use different state features). Using this method, a student agent has effectively improved learning both from an agent teacher [19] and from a human teacher [21]. The current work differs by using advice as the teaching approach rather than rules.

Learning from Demonstration [3] (LfD) is a paradigm in which a student agent watches another agent (or a human) and learns the teacher's policy. The main problem is one of generalization: how can the student learn to act in situations where it has not seen an explicit demonstration? In LfD, and the related approach of *imitation learning* [15], a student typically focuses on mimicking a teacher, not maximizing an external environmental reward. In contrast, our work assumes that the agent lives inside a well-defined MDP and should act to maximize a reward, with the help of a teacher.

A similar contrast applies to *inverse reinforcement learning* [1, 4, 13, 24] (IRL). Here, a student agent observes a teacher and tries to infer the teacher's reward function. Once estimating the teacher's reward function, the student autonomously learns to maximize it. IRL typically focuses on cases where the student cannot observe rewards, and the teacher is typically a human who has domain knowledge

about what constitutes a "good" policy. Our work assumes rewards are directly available to the student.

There is some work on teaching in non-RL settings, such as classification [7], or with collaborative teams of RL agents [17], whose problem settings differ somewhat from ours. There is also some recent theoretical work on when to teach a sequence of states and actions [23] and how to provide optimal demonstrations [5], but these approaches are currently limited to small, finite, MDPs. Lastly, there is a growing body of work that examines how a human *wants* to teach an agent in a sequential decision making task [5, 10, 11], and how an agent should be designed in order to take advantage of this insight; our work does not address human teachers.

7. Conclusions

As more problems become solvable by agent-based methods, it is important for agents to be able to work together, even if they are implemented differently. It is also important for agents and humans to be able to interact productively despite their substantial differences. RL agents are good at learning control policies for specific tasks, and it would be useful for them to be able to serve as teachers for those tasks.

This article poses the problem of having trained RL agents serve as teachers in ways that are effective for many types of students. We present teaching algorithms that use small amounts of action advice to speed up student learning, even when students learn and represent states differently. Our experimental results show that significant benefits, as measured by areas under student learning curves, are achievable with these algorithms.

There are many potential directions for future work. For example, action prediction might be improved by using an efficient incremental classifier. The concept of state importance could also use further exploration: perhaps there exist better domain-specific ways to measure state importance, or effective strategies for automatically selecting and adjusting importance thresholds.

Larger steps in future work could extend the problem to include multiple teachers and/or students. It would also be useful to examine agents with broader ranges of learning algorithms, including human students. Domains like Pac-Man and StarCraft would be particularly suitable for these kinds of experiments. It is our hope that this study will provide a solid foundation for additional work on these exciting questions.

8. Acknowledgments

The authors thank the reviewers for their comments and insights. This work was supported in part by NSF IIS-1149917.

References

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proc. of the International Conference on Machine Learning*, 2004.

[2] J. S. Albus. *Brains, Behavior, and Robotics*. Byte Books, Peterborough, NH, 1981.

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

[4] M. Babes-Vroman, V. Mari, K. Subramanian, and M. Littman. Apprenticeship learning about multiple intentions. In *Proceeding of International Conference on Machine Learning*, 2010.

[5] M. Cakmak and M. Lopes. Algorithmic and Human Teaching of Sequential Decision Tasks. In *AAAI Conference on Artificial Intelligence*, 2012.

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

[7] D. Chakraborty and S. Sen. Teaching new teammates. In *Proceedings of the Conference on Autonomous Agents and Multi Agent Systems*, 2006.

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

[9] T. Joachims. Making large-scale SVM learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

[10] F. Khan, X. J. Zhu, and B. Mutlu. How do humans teach: On curriculum learning and teaching dimension. In *Proceedings of Advances in Neural Information Processing Systems*, 2011.

[11] W. B. Knox, B. D. Glass, B. C. Love, W. T. Maddox, and P. Stone. How humans teach agents - a new experimental perspective. *International Journal of Social Robotics*, 4(4):409–421, 2012.

[12] L. J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[13] G. Neu. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2007.

[14] L. Nunes and E. Oliveira. On learning by exchanging advice. *AISB Journal*, 1(3), 2003.

[15] B. Price and C. Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

[16] P. Rohlfshagen and S. M. Lucas. Ms Pac-Man versus Ghost Team CEC 2011 competition. In *Proc. of the Congress on Evolutionary Computation*, 2011.

[17] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2010.

[18] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[19] M. E. Taylor and P. Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2007.

[20] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *J. of Machine Learning Research*, 10(1):1633–1685, 2009.

[21] M. E. Taylor, H. B. Suay, and S. Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2011.

[22] L. Torrey and M. E. Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2013.

[23] T. J. Walsh and S. Goschin. Dynamic teaching in sequential decision making environments. In *Conference on Uncertianty in Artificial Intelligence*, 2012.

[24] B. Ziebart , A. Maas, J. A. D. Bagnell, and A. Dey. Maximum entropy inverse reinforcement learning. In *Proceeding of AAAI Conference on Artificial Intelligence*, 2008.