Agents Teaching Agents: Preliminary Results for 1 vs. 1 Tactics in Starcraft

Nicholas Carboni
Computer Science Department
Lafayette College
carbonin@lafayette.edu

Matthew E. Taylor*
School of EECS
Washington State University
taylorm@eecs.wsu.edu

ABSTRACT

This paper describes the development and analysis of two algorithms designed to allow one agent, the teacher, to give advice to another agent, the student. These algorithms contribute to a family of algorithms designed to allow teaching with limited advice. We compare the ability of the student to learn using reinforcement learning with and without such advice. Experiments are conducted in the Starcraft domain, a challenging but appropriate domain for this type of research. Our results show that the time at which advice is given has a significant effect on the result of student learning and that agents with the best performance in a task may not always be the most effective teachers.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance

Keywords

Reinforcement Learning, Agent Teaching, Starcraft

1. INTRODUCTION

Reinforcement learning (RL) algorithms have become increasingly powerful, solving an impressive number of virtual and physical tasks [8, 22, 23]. However, the majority of work assumes that agents learn *tabula rasa*, even if another agent has already learned the task. Transfer learning [25] is one approach to solving this problem, allowing one agent to transfer knowledge to another agent in a similar task, but it typically relies on a deep understanding of the agents' internal representations. For example, an action-value function learned in one task could be used to speed up learning in a similar task. However, if the second agent has a different state representation or does not use temporal difference learning, this action-value function may be useless. If the agents do not have a shared robust communication protocol, e.g., they were made by different manufacturers, one agent may not understand what the other agent is trying to communicate. Finally, if the second agent is a human, a very different method of transfer would be required.

In this paper we consider the problem of a *teacher* agent teaching another *student* agent. The agents may use different learning algorithms and they may have different ways of representing the state of their environment. This is particularly important in light of 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. This work focuses on how one agent can provide action advice to another agent: 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. Action advice allows teaching with 1) limited compatibility requirements for the pairs of agents and 2) no dependence on the agent's underlying learning representation. We do not assume that the teacher is optimal — it is important that a student can learn to outperform its teacher, particularly if the student is more capable than the teacher. While this work does not consider a human in the loop, our long-term goal is to develop algorithms that will facilitate a productive relationship between an agent teacher and a human student. Teaching algorithms should thus have outputs that are human-understandable.

Our recent work [27] showed that agent-agent teaching was possible in the mountain car and pac-man domains. This paper focuses on Starcraft, a popular real time strategy domain that is significantly more complex than the past domains considered. We consider two novel advice algorithms, contributing to this family of algorithms designed to provide limited amounts of advice. Experiments compare the two algorithms and evaluate their ability to improve the performance of a student, relative to learning without a teacher. We also test how student performance changes with different teacher abilities.

Experiments are conducted in a 1 vs. 1 Starcraft battle. Results show that providing advice for just $\frac{1}{16}^{th}$ of the total training time can induce significant improvements to learning speed. Further, we show that student performance can depend on the ability of a teacher in a non-intuitive manner — an agent with high performance may not always be a better teacher than an agent with poorer performance.

2. BACKGROUND

This section provides a brief overview of background information necessary to understand our methods, discussed in Section 3.

2.1 Reinforcement Learning

Reinforcement learning agents execute actions in different environmental states, learning to maximize the expected long-term real-valued reward. On every step, the agent observes a state in the environment, $s \in S$. It then selects an available action, $a \in A$. The environment's transition function, $T : S \times A \mapsto S$, determines the next state the agent reaches, and the environment's reward function, $R : S \mapsto \mathbb{R}$, supplies the agent with the immediate reward. This work uses the common episodic setting, where learning is broken

*Much of this research was done while the second author was at Lafayette College.

into a series of independent learning traces. Each episode begins at a fixed start state, s_0, and ends in a set of final states. In our setting, a final state corresponds to one of the agents "winning" the battle.

Agents learn a control policy, $\pi : S \mapsto A$. A common way to represent such a policy is with a Q-function, $Q : S \times A \mapsto \mathbb{R}$, which estimates the total reward an agent will earn starting by taking action a in state s. An agent can maximize its rewards by always choosing the action with the maximal Q-value, assuming the Q-function is accurate. To improve the agent's estimate of Q, this work uses the common ϵ-greedy exploration approach, where the agent chooses a random action with a small probability ϵ (explore), and selects the estimated best action with probability $\epsilon - 1$ (exploit).

This work focuses on a task with a very large state space, making a tabular Q-function representation intractable. Instead, we will use tile coding, a function approximator that allows the agent to both generalize across nearby states, as well as distinguish between states with different Q-values [22]. In particular, there are a large set of features $\{f_1, f_2, ...\}$ which uniquely describe every state. The Q-function is a linear function of these weights, $Q(s, a) = \sum_i w_i f_i$, and learning an accurate Q-function reduces to learning accurate weights $\{w_1, w_2, ...\}$. For the experiments in this paper, we use Sarsa with tile coding function approximation. The tile coding is optimized pessimistically, as our previous work [27] found that pessimistic initialization outperformed optimistic initialization when providing advice to agents.

2.2 Teaching Agents

There is a growing body of work on improving RL agents by leveraging knowledge from other agents or humans. This section focuses on methods that work for both agent *and* human students. A more comprehensive discussion can be found in Section 6.

Learning from Demonstration [4] (LfD) is primarily used by the robotics community, but is also applicable to virtual robots. In this paradigm, a student agent will watch another agent or human and learn 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? LfD typically does not try to maximize an external environmental reward, which is often not present in real-world situations. In contrast, our work assumes that the agent acts inside a well defined MDP and should act to maximize a reward, not just to mimic a teacher.

A similar difference is observed in the *inverse reinforcement learning* [1, 5, 17, 28] (IRL) setting. 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. The teacher is typically a human who has domain knowledge about what constitutes a "good" policy and the student is an agent. Again, our work assumes rewards are available to the student.

Other 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 variables). Using this method, a student agent has effectively improved learning both from an agent teacher [24] and from a human teacher [26].

More similar are the ideas of *apprentice learning* [11], in which a student asks a teacher for advice whenever its confidence in a state is low, and *advice exchange* [18], in which peers ask for advice from each other based on heuristics of self-confidence and trust. Our work diverges from these by having an expert teacher decide when to give advice, and by focusing on the effective use of small amounts of advice (to help make the use of a human student feasible).

Algorithm 1 Early Episodes Advice

```
procedure EarlyEpisodesAdvice(π_s, π_t, n)
    for each each episode do
        for each visited state s visited by the student do
            if n > 0 then
                Advise π_t(s)
            else
                Follow π_s(s)
            end if
        end for
        n ← n − 1
    end for
end procedure
```

Unlike the previous methods, our previous work on agent teaching [27] interweaves teacher advice with autonomous student learning and focuses on the teacher deciding when to give advice. In particular, multiple methods were introduced for when a teacher should provide action advice to the student. This allowed the teacher to focus its advice on areas where the student performance was poor and/or the teacher was very confident, reducing the total amount of advice given. This reduction is important because it 1) allows the student to surpass the teacher's performance, and 2) is an important step towards minimizing the amount of advice needed to teach a student, which is critical if the student is an (impatient) human.

2.3 Learning in RTS Games

Real Time Strategy (RTS) games are a popular research platform due in part to their complexity and availability of highly-skilled human players. For instance, there have been multiple RTS tournaments created specifically for artificial agents [7]. There are many relevant subproblems for successful RTS play, including opponent modeling [12], planning build orders [10], and planning individual agent actions [6]. More recent work by Gemine *et al.* [13] considers a supervised learning approach to imitate observed trajectories.

More relevant to our paper are works that focus on reinforcement learning. For instance, Kresten *et al.* [3] and Marthi *et al.* [16] look at both base building and controlling individual units by taking a hierarchical RL approach. This is in contrast to the current work, where we focus on the more simple task of only controlling units. Other work [20] considers only controlling the movement of agents, but focuses on transfer between different scenarios through a case based reasoning mechanism. In contrast, our paper focuses on transferring knowledge not between similar agents in different tasks, but between different agents in the same task.

3. TEACHING ALGORITHMS

This work investigates two different algorithms that allow one agent (the teacher) to provide advice to another agent (the student). Both the student and the teacher know the current state of the student. The teacher then decides if it should tell the student what action to take. We assume that the student always executes an action suggested by the teacher when it is given. Other desired objectives are to 1) allow improvements to learning with relatively little advice and 2) allow the student to outperform the teacher.

3.1 Early Episodes Advice

The Early Episodes Advice algorithm (Algorithm 1) has the teacher provide all advice to the student as quickly as possible by telling

Algorithm 2 Alternating Advice

```
procedure AlternatingAdvice(π_s, π_t, ep_s, ep_t, n)
    episode ← 0
    for each each episode do
        if episode = ep_s + ep_t then
            episode ← 0
        end if
        for each state s visited by the student do
            if episode < ep_t & n > 0 then
                Advise π_t(s)
            else
                Follow π_s(s)
            end if
        end for
        episode ← episode + 1
        n ← n − 1
    end for
end procedure
```

the student what action to take, according to its learned policy π_t, for the first n episodes. This algorithm leverages the intuition that advice is more valuable earlier in the learning process when the student knows little about its domain. Also, getting information earlier allows the the student to use that knowledge for a longer time and possibly gain a larger total reward. The downside to this algorithm is that the student agent is only given advice about a small percentage of the state space (i.e., only those states visited when following the teacher's advice). Thus, the student will be directed to a goal state early in learning, but it will not have experience in states not visited by the teacher's policy when it must follow its own policy, π_s.

3.2 Alternating Advice

The shortcomings of Early Episodes Advice lead us to develop an additional algorithm, Alternating Advice, which allows the student agent to experience more of the state space than is covered by the teacher's policy. To do this, Algorithm 2 alternates episodes where the teacher provides advice with episodes in which the teacher provides no advice. The teacher again provides a total of n episodes of advice. Now, however, the teacher will provide advice for all the states in ep_t episodes, according to the teacher's policy π_t, and then the student will follow its own policy for ep_s episodes, using π_s. This way the student is allowed to explore more of the state space in the context of the advice it was given before the teacher exhausts the amount of advice it was allotted. Although this algorithm gives the student more wide ranging knowledge of the domain, it will likely drift away from the optimal trajectory.

4. STARCRAFT

This section provides a brief introduction to Starcraft and the details necessary to recreate the experimental results discussed in the following section.

4.1 The Full Game of Starcraft

Starcraft is an RTS game in which human and/or computer players work against each other to destroy the opposing player's structures and units. The traditional game consists of three playable races; each of which comes with their own set of units and structures. Players must 1) collect resources which can be spent on 2) creating buildings, 3) creating units, and 4) upgrading unit capabilities. As units are created, the players must explore the world and attempt to eliminate the other teams. A diverse set of skills, including resource management, build order, upgrade order, exploration, offensive and defense strategy, and small-scale tactics are required for successful play at the highest levels.

4.2 Starcraft Experimental Domain

The full game of Starcraft is beyond the scope of the current work. Instead, we focus on one-on-one tactics with two common units. Figure 1 shows the center of the board used in experiments. 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; it must be within close range of its enemy to attack.

This map consists of one island on which the two previously mentioned units fight. There is also a barrier between the two units through which they can not pass.

4.3 Learning in Starcraft

Our goal is to have the Marine learn to consistently kill the Zergling. The Zergling is controlled by the standard game AI: it is stationary until the Marine is close enough for it to see it, or if the Marine shoots at it. A trial in our experiment consists of 800 total episodes which alternate between learning episodes and test episodes in which learning and teaching is disabled in order to examine the agent's progress. Each episode begins from the fixed start state. The episode ends when one of the units dies, or a maximum of 1000 actions have been executed by the agent.[1]

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

The agent represents the current state using six state variables: the agent's X and Y position, the straight line distance to the enemy, the difference in hit points between the agent and the enemy, a boolean value for whether the enemy is stationary, and the angle of the enemy relative to the agent.

The agent receives a reward of -0.3 on every step. 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 receives a final reward of -300 because it has executed 1000 actions, each of which has a small 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 case results in a reward of roughly -15. In the best case scenario our agent will kill the enemy from behind cover without being hit. This scenario would result in an episode reward of roughly 20.

In any state the agent can execute one of seven actions. For one time step the agent can stop, attack the enemy, move towards the enemy, move south, move north, move east, or move west. The attack command will cause the Marine to shoot the enemy if it is

[1]In our implementation, if 1000 actions have been executed, this typically means that the Marine is exploring the state space far away from the Zergling. If the maximum number of actions is exceeded, further learning is disabled for the remainder of the episode and the Marine runs to the Zergling without firing, killing itself to end the episode.

Figure 1: This screenshot shows the Starcraft map used in experiments. A Marine and Zergling have fixed start positions. If the Marine rushes at the Zergling, he will lose the encounter. However, if he begins attacking from cover, he will survive.

within range, or move towards the enemy if is not. Recall that 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, making it particularly difficult for the agent to learn to correctly attack the enemy.

The agent learns using Sarsa with a fixed exploration rate of 0.1 and a fixed learning rate of 0.1. A CMAC [2] tile coding is used for function approximation, where each of the 6 state variables are tiled independently with 32 tilings.

Starcraft, like most similar strategy games uses fog of war to deliberately obscure units and structures from the players' views. To simplify our experiments, the fog of war was disabled at the start of experiments. (If the fog of war is enabled, and the Marine has not yet moved close enough to the Zergling, the actions for attacking the enemy and moving towards the enemy have no effect.)

5. EXPERIMENTAL RESULTS

Figure 2 shows the results of learning in this Starcraft task. All learning curves are averaged over 20 trials with a 10-episode moving window. Each trial was 800 episodes long. On even numbered episodes, the agent was allowed to learn (with or without advice). Odd numbered episodes were test episodes, in which learning was disabled. If training episodes were graphed, they would show near-perfect performance on episodes in which the agent received advice from a good teacher, and similar performance to the test episodes if no advice was received. Graphs in this section show only the performance on test episodes.

The No Teaching line shows the performance of the agent learning without any prior knowledge. This is the baseline performance of the Sarsa algorithm. Each trial took approximately 30 minutes on a 2.5 GHz laptop running Windows Vista. Standard error bars over 20 trials are shown every 10 episodes to give an idea of the noise inherent in evaluating policies in this domain.

5.1 Using a Good Teacher

The Early Episodes Advice algorithm gives all advice at the beginning of each trial. We chose $n = 25$, so that the learning agent was given teacher advice for the first 25 episodes. We expected the student to use the advice it was given early on to inform its future actions therefore making learning easier.

The Alternating Advice algorithm was also allowed to give advice for 25 episodes. However, it spreads its advice out more, allowing the agent to interweave its own unassisted learning. The line "Alternating Advice, 1" set $ep_s = ep_t = 1$, so that advice was given every other episode. The line "Alternating Advice, 5" instead alternated gave advice for 5 episodes with 5 episodes of autonomous learning, up to a total of 25 episodes of advice. In both cases, after all the advice was provided, the student was left to learn on its own for the remainder of the trial.

The dotted horizontal line in Figure 2 shows the performance of the teaching agent, which had been trained for 500 episodes. The three experiments that use this teacher all outperform the "No Teaching" benchmark, showing that advice does indeed improve performance.

The three teaching algorithms begin with roughly the same performance. Recall that only test episodes are graphed — if the training episodes were graphed, where the student followed the teacher's advice, the performance would be similar to the Average Teacher Performance line. After 40 episodes, the average performance of all of the teaching-enabled methods are outperforming the average performance of learning without teachers. This average relative performance improvement continues through the end of the trial, although we expect all methods to converge in the limit. Counter to our expectations, none of the three teaching methods dominated the other two. One notable difference is that the performance of the student using Early Episodes Advice, does drop significantly around episode 25, where the advice has been exhausted, whereas the two Alternating Advice methods do not suffer this drop.

On average all of these algorithms lead the Marine to consistantly kill the Zergling as we had hoped, from behind cover before it was wounded.

5.2 Using a Poor Teacher

To study the effect of teacher quality on our algorithms, we allowed an agent to learn for only 250 episodes, resulting in a policy whose average performance is graphed in Figure 3 as "Average Teacher Performance." Because this teacher's performance is significantly worse than the previous teacher, we expected student performance to significantly decrease. Surprisingly, we find that *the initial performance of the student is increased*, relative to the good teacher. The performance of the algorithms using the poor teacher is better than the algorithms using the good teacher for the first 100 episodes. After the first 100 episodes, the performance of the students in Figure 2) outperform those of the poor teacher.

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 actually reach a much 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.

This behavior exposes the fact that different teachers may help agents optimize different types of learning improvement. Depending on the scenario, initial performance in a new task may be critical. For instance, in a robotic task, it may be extremely important to quickly learn to execute actions that do not harm the robot. On the other hand, the total reward or the final reward may be the most im-

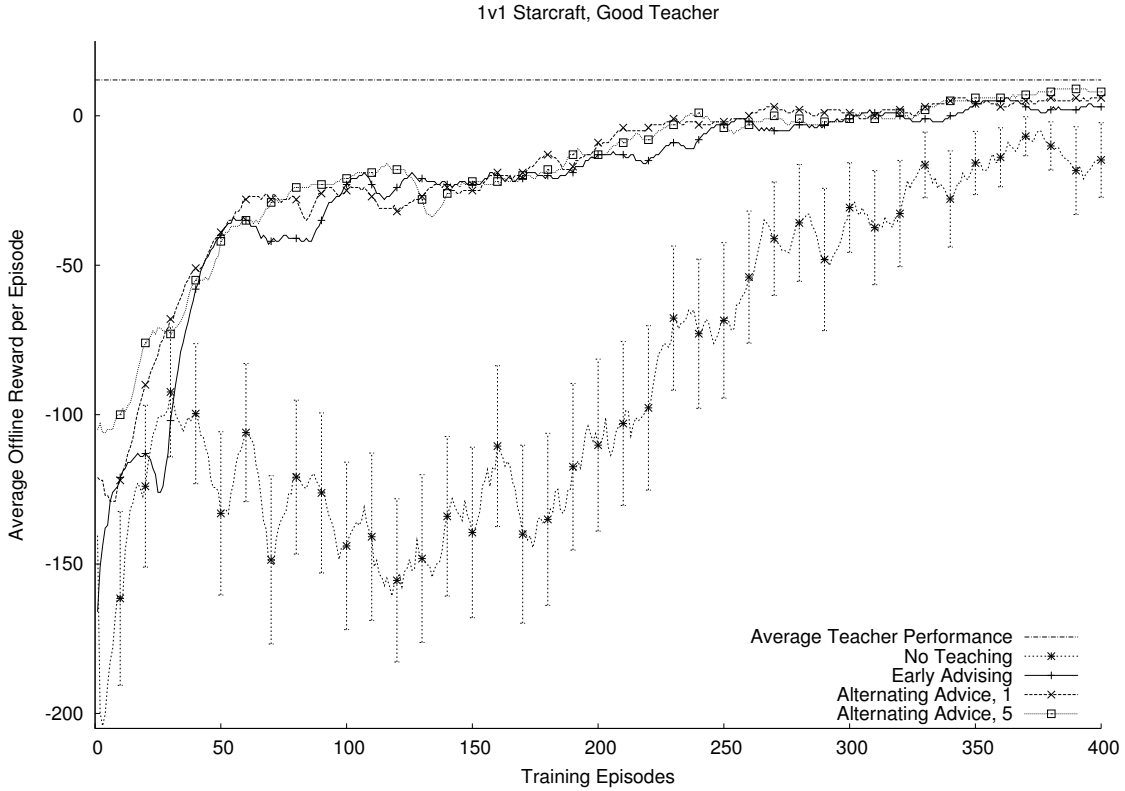

Figure 2: This graph shows the performance of different learning algorithms. The x-axis shows how many training episodes have passed. The y-axis is the average reward of the training method, tested with learning disabled. The No Teaching line is the baseline performance of Sarsa. The Average Teacher Performance shows the average performance of the teacher, used by the three learning methods that make use of a teacher, which all outperform learning without teaching.

portant metric to maximize. We currently have little intuition as to how to best pick teachers to maximize different speed-up metrics, but this will be an important question for future work.

6. ADDITIONAL RELATED WORK

There are many possible ways for agents to help agents learn, but few 1) are also applicable to human students and 2) maximize an environmental reward rather than mimicking a teacher. This section considers other methods for helping agent learning that are not directly applicable to the scenarios discussed in this paper.

Imitation learning [19] allows a student to learn by observing a teacher, using supervised learning to attempt to mimic the teacher. It is related to *Learning from Demonstration* [4], where agents learning to mimic a (typically, human) demonstrator.

There are several types of related work in the area of helping agents learn. Some of this work involves teaching in non-RL settings, such as classification [9], or involves collaborative teams of RL agents [21]. These areas of research address the same high-level goal of productive agent interaction, but their problem settings are somewhat different.

More closely related work has one RL agent teach another. For example, in *experience replay* [14], a student trains on the recorded experiences of a teacher. This requires the student to have an identical state representation, which is a limitation our methods avoid.

Lastly, there has also been work on allowing humans to communicate rules expressing their knowledge of a domain [15]. Research in these areas tends to focus on compensating for human error, whereas we wish to design our methods to account for suboptimal teachers.

7. CONCLUSIONS AND FUTURE WORK

Our experiments showed that the three types of advice produced improvement in student learning, while the performance of the student learning with Early Episodes Advice is initially lower than that of the Alternating Advice algorithm.

From these results we can conclude that both Early Episodes Advice and Alternating Advice are effective algorithms for teaching an agent a complex task in a teacher-student framework. Alternating Advice proved to be slightly more effective than Early Episodes Advice. We believe this advantage stems from the student gaining a more comprehensive knowledge of the domain earlier than it would in Early Episodes Advice.

Future work will change our algorithms to alternate per action, rather than per episode, and to attempt to further integrate student exploration with teacher advice. We would like to study how different types of teacher behaviors affect student performance and test whether the benefits of teaching can be predicated, or if different teachers can be combined to teach a single student. Lastly, these teaching algorithms should be tested on human students and with more than two agents in a task, for which purposes Starcraft is a very attractive domain.

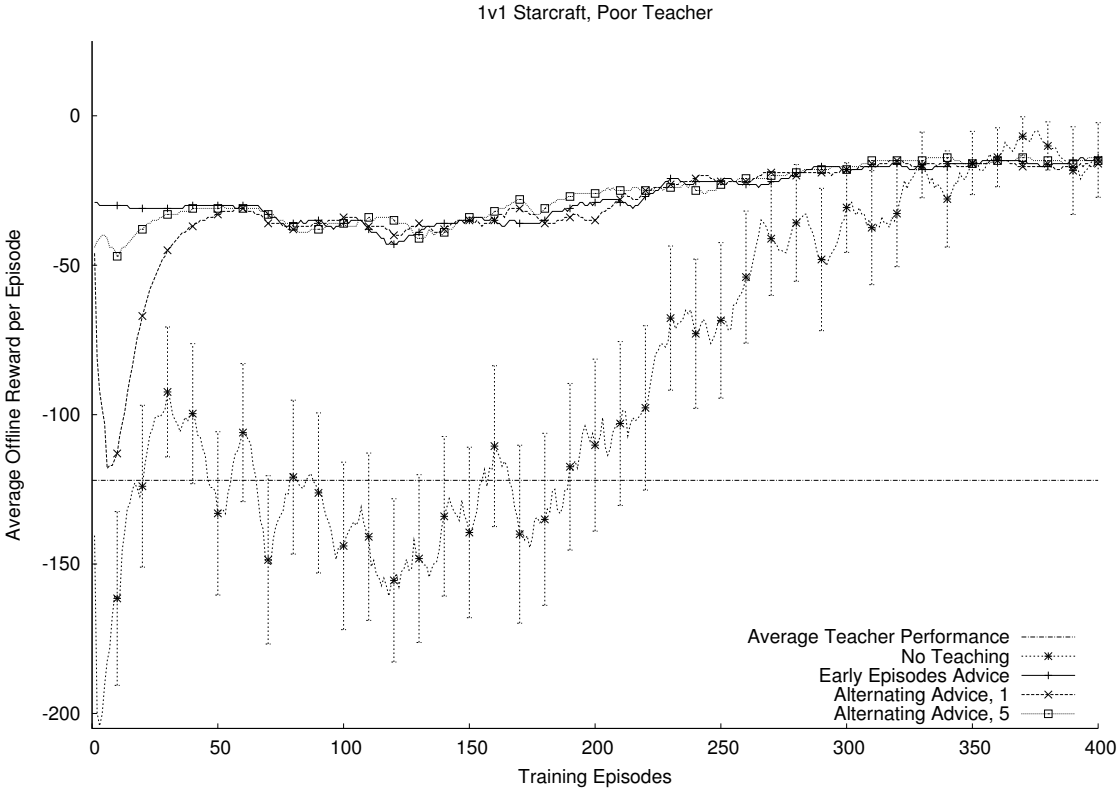

Figure 3: This graph shows the performance of the same algorithms as in Figure 2. However, now the teacher is significantly worse than in the previous set of experiments. Surprisingly, agents using this poor teacher are initially better than agents in the previous figure, but this advantage is reversed after the first 100 episodes. An average reward greater than zero means the student, on average, kill the enemy more often than not.

8. ACKNOWLEDGMENTS

The authors thank Lisa Torrey for her helpful comments and suggestions. This work was supported in part by NSF IIS-1149917.

9. REFERENCES

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 2004.

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

[3] K. T. Andersen, Y. Zeng, D. D. Christensen, and D. Tran. Experiments with online reinforcement learning in real-time strategy games. *Applied Artificial Intelligence*, 23(9):855–871, 2009.

[4] 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.

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

[6] R.-K. Balla and A. Fern. UCT for tactical assault planning in real-time strategy games. In C. Boutilier, editor, *IJCAI*, pages 40–45, 2009.

[7] M. Buro and D. Churchill. Real-time strategy game competitions. *AI Magazine*, 33(3):106–108, 2012.

[8] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, 2010.

[9] D. Chakraborty and S. Sen. Teaching new teammates. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 691–693, 2006.

[10] D. Churchill and M. Buro. Build order optimization in starcraft. In *AIIDE*, 2011.

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

[12] E. W. Dereszynski, J. Hostetler, A. Fern, T. G. Dietterich, T.-T. Hoang, and M. Udarbe. Learning probabilistic behavior models in real-time strategy games. In *AIIDE*, 2011.

[13] Q. Gemine, F. Safadi, R. Fonteneau, and D. Ernst. Imitative learning for real-time strategy games. In *CIG*, pages 424–429. IEEE, 2012.

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

[15] R. Maclin and J. W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22(1-3):251–281, 1996.

[16] B. Marthi, S. Russell, D. Latham, and C. Guestrin. Concurrent hierarchical reinforcement learning. In *Proceedings of the 19th international joint conference on Artificial intelligence*, IJCAI'05, pages 779–785, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

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

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

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

[20] M. Sharma, M. Holmes, J. Santamaria, A. Irani, C. Isbell, and A. Ram. Transfer learning in real-time strategy games using hybrid CBR/RL. In *In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.

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

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

[23] C. Szepesvári. *Algorithms for Reinforcement Learning*. Morgan and Claypool, 2009.

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

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

[26] 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 (AAMAS)*, May 2011. 22

[27] 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 (AAMAS)*, May 2013.

[28] B. Ziebart , A. Maas, J. A. D. Bagnell, and A. Dey. Maximum entropy inverse reinforcement learning. In *Proceeding of 23rd AAAI Conference on AI*, July 2008.