Coaching Advice and Adaptation

Patrick Riley and Manuela Veloso

Carnegie Mellon University, Computer Science Department, Pittsburgh, PA 15232
{pfr,mmv}@cs.cmu.edu
http://www.cs.cmu.edu/~{pfr,mmv}

Abstract. Our research on coaching refers to one autonomous agent providing advice to another autonomous agent about how to act. In past work, we dealt with advice-receiving agents with fixed strategies, and we now consider agents which are learning. Further, we consider agents which have various limitations, with the hypothesis that if the coach adapts its advice to those limitations, more effective learning will result. In this work, we systematically explore the effect of various limitations upon the effectiveness of the coach's advice. We state the two learning problems faced by the coach and the coached agents, and empirically study these problems in a predator-prey environment. The coach has access to optimal policies for the environment, and advises the predator on which actions to take. We experiment with limitations on the predator agent's actions, the bandwidth between the coach and agent, and the memory size of the agent. We analyze the results which show that coaching can improve agent performance in the face of all these limitations.

1 Introduction

In spite of the increasing complexity of the relationships among autonomous agents, one agent coaching or advising another is still somewhat uncommon. We frequently see this relationship among human beings, yet further computational understanding of how this relationship extends to autonomous agents is needed.

As we start to move beyond agent systems of short, limited duration, we believe that advising relationships will become more important. Many current agent systems implicitly allow a complete transfer of knowledge of an agent's behavior structure, both because of the size of the domains and the homogeneity of the agent's representations of the world. In other words, if one agent knows a good way to behave in the world, all agents can easily copy that knowledge. A coach relationship does not assume this is possible. The goal is to obtain knowledge transfer in spite of limitations in communication bandwidth or differences in behavior representation. If the coach wants the agent to act differently, the coach must still consider the best way to *guide* the agents to this behavior, rather than simply transferring the knowledge. Additionally, a coach should adapt its idea of optimal behavior to the strengths and abilities of the agent being coached.

Specifically, our research on coaching refers to one autonomous agent providing advice to another about how to act in the world. Through research on a complex simulated robot soccer domain [1, 2], we have been exploring how one agent can generate advice for teams. With this experience, we have now undertaken a series of systematic experiments in more controlled environments. We present a general scheme for how the coach and agent interact in Figure 1. The coach can only affect the world through communication with the agent. Both the coach and the agent are learning. This interaction between the coach and agent is challenging, as the coach has only an indirect effect on the world and does not get to see the working of the agent's decision mechanism. We empirically investigate varying the limitations of the agent as well as limitations of the communication between the coach and agent.

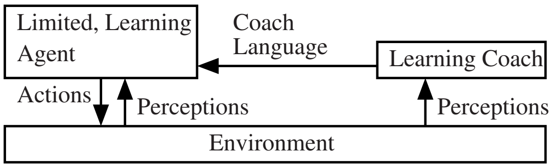

Fig. 1. The high-level view of coach-agent-environment interaction

While our previous work has focused on adapting to an adversary and compiling past experience, the research here deals primarily with adapting advice to the agent being coached. One hypothesis that underlies our current work is that in order for a coach to be effective coaching multiple different agents, it needs to adapt its advice to the peculiarities of each agent. This is supported by the results of the coach competition at RoboCup 2001 as discussed in [1, 3] where the ability of a coach to improve a team's performance varied vastly across different teams. This paper contributes interesting interaction modes of a coach and coached agent and a thorough empirical exploration of the ramifications of various limitations on that interaction. We start from the idea that advice is recommending an action for a particular state and explore the effects in different interaction modes. We choose basic reinforcement learning algorithms as a starting point since our interest is in the comparative performance of different algorithms and what that suggests for the general coaching problem.

2 Related Work

This work focuses primarily on how an advice giver can interact with an advice taker, and how the characteristics of that interaction affect the performance of the agent. One important component of this is how the advice-taker will function. A variety of work has been done on using advice (often from humans) to improve autonomous agent performance.

A significant challenge is to operationalize "fuzzy" advice. For example, [4] describes a multi-step method to incorporate advice into a reinforcement learner which uses a neural network to represent the value function. [5] takes a similar approach. High level advice from a human being is translated to if-then rules describing intermediate goals for the agents. By combining these with qualitative descriptions of the domain dynamics, these rules can then be refined with a genetic algorithm. [6] presents a model for advice taking in a recurrent connectionist network. Advice changes the current activation state rather than the weights of the network.

Clouse and Utgoff [7, 8] take a similar approach to ours to study the effect of advice on a reinforcement learner. Their focus is primarily on the advice-taker and how to incorporate advice into the value updates, and little attention is paid on how the advice was generated in the first place. They do empirically explore how the rate of advice affects performance. However, they do not allow the training agent to give advice for states other than the current state or explore limiting the memory of the agent. Most of the past work on autonomous agents *giving* advice has been in tutoring systems.

A closely related research area is that of imitation (e.g. [9]). Similar work has gone under many different names: learning by demonstration. behavior cloning, learning by watching. and behavior or agent imitation. In all the cases, the robot or agent is given examples of some task being done successfully (often from a human demonstration), and the agent's goal is to perform the same task. The demonstration of the task can be seen as a set of advised actions for the agent.

3 Continual Advice

We first consider an agent without limitation and where the coach can communicate some advice every step with the agent. The first question which must be answered is how an advice-receiving agent incorporates advice into its behavior.

3.1 Learning Algorithms

We begin with a basic Q-learning agent. The state space is assumed to be represented explicitly.

During learning, the agent has a fixed exploration probability ϵ; with probability ϵ the agent takes a random action and with probability $(1 - \epsilon)$ it chooses uniformly randomly between the actions with the highest Q-value. This is commonly known as ϵ-greedy exploration.

For all Q-learning done throughout, the learning rate decays over time. For each Q-table cell, we keep track of the number of times that cell has been updated (call it v), and calculate the learning rate for each update as $\frac{1}{1+v}$.

Then we add a coach agent. Every step, the coach recommends a single action. Table 1 shows the algorithm used by the coached agent. The only difference with a basic Q-learner is in choosing an action. The new parameter β controls the probability that the agent will follow the coach's advice. Otherwise, the agent reverts to the normal exploration mode. Except where noted, we use $\beta = 0.9$.

The coach is also a Q learner, but its actions are to advise *optimal* actions. The algorithm for the coach is shown in Table 2. While this algorithm is apparently too complex for this particular task, we will continue to use the same

Table 1. Algorithm for an agent Q-learning and receiving continual advice

```
ContinualAdviceTaker($\epsilon$, $\beta$)
  $a$ := recommended action
  with prob $\beta$
    do $a$
  with prob $1 - \beta$
    Choose action ($\epsilon$ greedy)
  Q-update for action performed
```

Table 2. Algorithm for the coach Q-learning and providing advice

```
CoachContinualAdvice($\epsilon$)
  $g$ := last rec. (a.k.a guidance)
  if (see agent action (call it $a$))
    if ($a$ is optimal)
      Q-update for $a$
    else
      no Q-update
  else
    Q-update for $g$
  recommend action ($\epsilon$ greedy)
```

algorithm later in more complex settings. There are two primary cases (which we vary as an independent variable): seeing the agent's actions and not seeing the agent's action. If the coach sees the actions and the agent takes an optimal action, the coach performs a Q-update with that action. This is based on the assumption that if the coach had advised that action, the agent probably would have done it. Note that in general, there may be multiple optimal actions in a particular state. However, if the action is not optimal, the coach does nothing; the coach's Q-table does not even include non-optimal actions.

Note that this algorithm requires that the coach know all optimal actions. For these experiments, we precompute the optimals beforehand.

What the coach is learning here (and will be learning throughout) is not what the optimal actions are (the coach already knows this), but rather learning about what actions the agent is taking. The coach restricts its Q-table to just the optimal actions and then pessimistically initializes the table. The Q-table then provide an estimate of the value achieved by the agent when the coach *recommends* an action (which is not the same as an agent *taking* the action). This will have important consequences when the advisee agent has limitations in the following sections.

3.2 Experimental Results in Predator-Prey

We use a predator-prey environment in order to test all the algorithms. The simulation is built upon the SPADES simulation system [10]. The agents operate on a discrete 6x6 grid world. The agents have 5 actions, stay still or move north, south, east, or west. There are virtual walls on the outside of the grid so if an agent tries to move outside the grid, it stays where it is.

There are *two* prey agents which, for these experiments, move randomly. The predator agent's goal is to capture at least one prey agent by being on the same square as it. The coach's job is to advise the predator agent. The world is discrete, all agents have a global view of the world, and all moves take place simultaneously.

The world is operated episodically, with an episode ending when at least one prey is captured; the predator can simultaneously capture both prey if all three agents occupy the same square. The predator and coach receive a reward of 100 for each prey captured and a reward of -1 every step that is taken. The agents try to maximize *undiscounted* reward per episode.

The state space of the world is $36^3 = 46656$ (the predator location, prey 1 location, prey 2 location), though for any state which is a capture for the predator (2556 states), there are no actions so no learning need take place. After a capture, all agents are placed randomly on the grid. Further note that approximately 20% of the states in this environment have multiple optimal actions.

The optimal average reward per step is approximately 17, which means on average between 5 and 6 steps to capture a prey. The average reward per step reflects the value of a policy since we are using episodic undiscounted reward. In all cases, the coach has access to the complete true values of all actions in all states and therefore optimal policies of the environment.

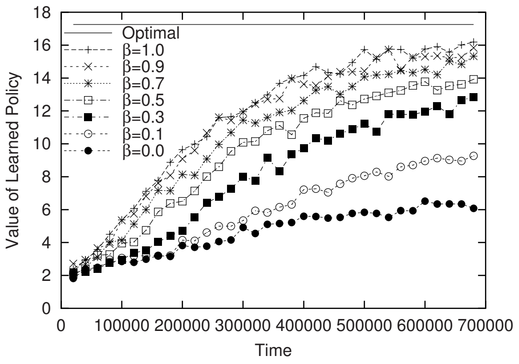

Fig. 2. Data for a predator agent learning with a coach advising the agent every cycle

We alternated periods of learning and evaluation. Every 5000 steps, 5000 steps of policy evaluation were done. The results throughout are the average values per step obtained during the policy evaluation periods. In order to smooth the curves and make them more readable, the values of two periods of policy evaluation were averaged together before plotting.

Throughout, we used $\epsilon = 0.1$ as the exploration parameter. The Q table was initialized to 0 everywhere, and the optimal Q values are positive everywhere.

The $\beta = 0.0$ line in Figure 2 shows the results for the predator agent learning without the coach. The agent is learning, though over 700,000 steps (half of which are learning, half are evaluation), the agent achieves only about 40% of optimal.

The rest of Figure 2 presents the results of the coach advising (with the algorithm CoachContinualAdvice as shown in Table 2) and the predator taking advice (with the algorithm ContinualAdviceTaker as shown in Table 1), varying the value of β to the ContinualAdviceTaker algorithm. This value β controls how often the predator listens to the coach, and how often the predator ignores the advice. As expected, with the coach, the predator agent learns much more quickly and reaches nearly an optimal policy. The coach is effectively guiding the agent to the right actions, which improves the speed at which the predator's Q table reflects an optimal policy. Also, the more often the predator listens to the coach (i.e. as β increases), the faster the learning and the better the performance. However, there are diminishing returns with the performance difference between $\beta = 0.7$ and $\beta = 0.9$ rather slight and the difference between $\beta = 0.9$ and $\beta = 1.0$ not significantly different.

4 Limited Agent Actions

We now consider cases where the action space of the coached agent is limited. For our purposes, this will simply mean that the agent is not allowed/able to perform some actions.

4.1 Learning Algorithms

A revised algorithm for the coached agent is shown in Table 3. The only difference from ContinualAdviceTaker (Table 1) is that if the coach recommends an action the agent can not perform, a random action is performed. This is intended to simulate an agent which is not fully aware of its own limitations. By trying to do something which it can't do, some other action will result.

Table 3. Algorithm for a predator with limited actions. Note that for the random action done, the same action is chosen every time that state is visited

```
LimitedContinualAdviceTaker($\epsilon$, $\beta$)
  $a$ := recommended action
  with prob $\beta$
    if ($a$ is enabled)
      do $a$
    else
      do random enabled action
  with prob $1 - \beta$
    Choose action with $\epsilon$ greedy exploration
  Q-update based on action performed
```

The coach can follow the same CoachContinualAdvice algorithm as before (Table 2) and should be able to learn which optimal policy the agent can follow. If the coach can see the agent's actions, then only those Q-values for actions which the agent can perform will have their values increased. If the coach can not see the actions, then recommending an action that the agent can perform will tend to lead to a higher value state than recommending an action the agent can not perform (since the agent then acts randomly). In this manner, the coach's Q-table should reflect the optimal policy that the agent can actually perform, and over time the coach should recommend less actions which the predator agent can not do. Note that this algorithm implicitly assumes that the agent can perform some optimal policy.

4.2 Experimental Results in Predator-Prey

We once again use the predator-prey world as described in Section 3.2.

We assume that the limitations on the agent still allow some optimal policy. In particular, in this environment, 9392 of the 46656 states have more than one optimal action. Every choice of optimal action for each of those states gives an optimal policy, and we restrict the predator agent to be able to perform exactly one of the optimal policies. The same restriction was used for all experiments that are directly compared, but different experiments have different randomly chosen restrictions.

Further, for a given state action pair, the same random action always results when the agent is told to do an action it can not do. We also chose to use $\beta = 1.0$ to emphasize the effects of the coach's advice.

Results of this learning experiment are shown in Figure 3. An important issue to consider when analyzing the results is how much our limitation algorithm actually limits the agent. Only 20% of the states have multiple optimal actions, and of those, almost all have exactly 2 optimal actions. If an agent performs optimally on the 80% of the state space with one optimal action and randomly anytime there is more than one optimal action, the average reward per step of this policy is approximately 14.2 (optimal is 17.3).

In order to provide a reasonable point of comparison, we also ran an experiment with the coach providing intentionally bad advice. That is, for every state where the predator has an optimal action disabled, the coach would always recommend that action, and the predator would always take the same random action in a given state. This is the data line "Always Rec. Disabled" in Figure 3. The data line for the predator learning without limitations and with the coach (with $\beta = 0.9$ from Section 3) is shown labeled "No Limitation." Whether or not the coach sees the actions, the learning coach achieves better performance than the baseline of bad advice and approaches the performance without limitations at all.

A natural question to ask at this point is whether the coach is really learning anything about the agent's abilities. One measure of what the coach has learned about the agent is to examine what percentage of the coach's recommended actions are ones that the agent can not do. Figure 4 shows this value as the simulation progresses. The three lines represent the case where the coach is not learning (basically flat), learning and not seeing the agents actions, and learning and seeing the agent's actions. As one would expect, with learning, the percentage of bad actions recommended goes down over time, and seeing the actions enables faster learning.

5 Limited Bandwidth

Up to this point, the coach was giving advice to the agent every cycle. We see this as an unrealistic interaction mode with the coach, and this set of experiments

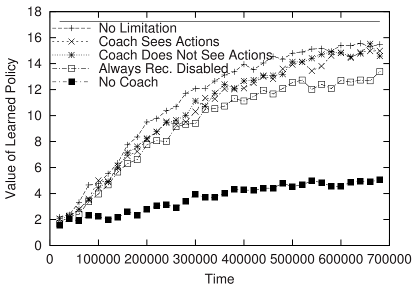

Fig. 3. The value of the policy learned by a limited predator agent under various coach conditions

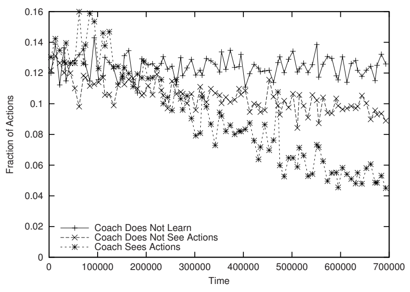

Fig. 4. As a function of time, the percentage of coach recommended actions which the predator can not do

Table 4. RandomState strategy for coach giving advice with limited bandwidth

```
CoachRandomState()
  if (time for $K$ more advice)
    do $K$ times
      $s :=$ random state
      $a :=$ random optimal action for $s$
      advise $(s, a)$
```

deals with limiting the amount of advice provided, while still using an advisee agent with a limited action space (as described in Section 4).

5.1 Learning Algorithms

First, to allow the coach to talk about states that are not the current state, a single piece of advice is now a state-action pair. The coach is advising an agent to perform a particular action in a particular state. Further, the coach has limitations on how much advice it can give. We use two parameters: I is the interval for communication and K is the number of states that the coach can advise about. Every I cycles, the coach can send K pieces of advice.

The agent stores all advice received from the coach and consults that table in each state. The new algorithm is shown in Table 5. For this experiment, the table T simply stores all advice received and if multiple actions have been advised for a given state, the table returns the first one received which the agent is capable of doing.

We propose two strategies for sending advice. The first strategy is mostly random; the coach randomly chooses K states and sends advice for an *optimal* action for each of those states (see Table 4). Note that while we call this a random strategy, it is still providing optimal advice about the states it chooses. If the bandwidth between the coach and advice taker were large enough, RandomState would send the entire optimal policy to the agent.

Table 5. Algorithm for a predator Q-learner with limited bandwidth with the coach; T is a table which stores past advice from the coach

```
LimitedBWAdviceTaker($\epsilon$, $\beta$, $T$)
    if (advice received from coach)
        add advice to $T$
    if (current state is in $T$)
        $a$ := rec. action from $T$
        with prob $\beta$
            do $a$
        with prob $1 - \beta$
            Choose action ($\epsilon$ greedy)
    else
        Choose action ($\epsilon$ greedy)
    Q-update for action performed
```

Table 6. OptQ strategy for coach giving advice with limited bandwidth

```
CoachOptQ($\epsilon$)
    $Q^*$ := optimal Q-table
    $V^*$ := optimal value function
    every cycle
        $s$ := current state
        $a$ := action predator took
        put $(s, V^*(s) - Q^*(s, a))$ in $W$
        if ($a$ was an advised action for $s$)
            perform Q-update
        if (time to send $K$ advice pieces)
            $W' = $ (smallest $K$ values of $W$)
            for each $(s', x)$ in $W'$
                if ($x \approx 0$)
                    $s'$ := random state
                $a'$ := action for $s'$ ($\epsilon$ greedy)
                advise $(s', a')$
```

The other strategy, which we call "OptQ" is more knowledge intensive. It requires the entire optimal Q table and always seeing the coached agent's actions. The algorithm is given in Table 6. Like RandomState, OptQ always provides optimal advice for the states it chooses. The difference is that OptQ attempts to choose better states about which to advise. The basic idea is to advise about the states in the last interval for which the agent performed the least optimal actions. Note that the *smallest* values are chosen first since all of the values in W are negative. If the algorithm runs out of states about which to advise, it simply chooses random ones rather than wasting the bandwidth.

While neither RandomState nor OptQ may be good algorithms to implements in a real world setting, they provide good bounds on the range of performance that might be observed. RandomState puts no intelligence into choosing what states to advise about, and OptQ uses more information than would likely be available in any real world setting. The improvement that OptQ achieves over RandomState indicates how much benefit could be achieved by doing smarter state selection in an advice giving framework like this one.

5.2 Experimental Results in Predator-Prey

We once again use the predator prey world as described in Section 3.2. The predator has limited actions as described in Section 4.2. For the parameters limiting advice bandwidth, we chose I to be 500 and K to vary between 1 and 50. Recall that every I cycles, the coach can send K pieces of advice.

The results are shown in Figure 5 for different values of K. Note first the that as the amount of information the coach can send gets larger (i.e. K gets

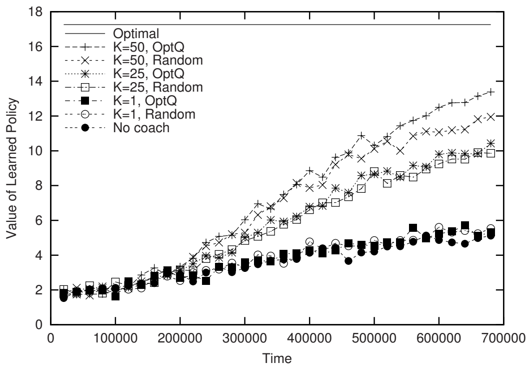

Fig. 5. Policy evaluation for the predator for various coach strategies and values of K

larger), the agent learns more quickly. While the same performance as the coach advising every cycle is not achieved, we do approach that performance. However, it should be noted that the agent is remembering all advice given.

The surprising result is that the OptQ algorithm with its much higher information requirements does not perform much better than the RandomState strategy. There are two things to consider. First, by the end of the simulation, the RandomState strategy achieves fairly large coverage of the state space. For example, with $K = 25$, the random strategy is expected to provide advice about 47% of the states[1]. Since the agent remembers all of the advice, near the end of the run it is essentially getting optimal advice for one in two states. Secondly, the OptQ strategy only advises about states about which the agent has already been and taken an action. The chance of encountering one of these states again is about the same as encountering any other given state.

Differences between RandomState and OptQ performance only emerge at the $K = 50$ level. This suggests as the bandwidth of the advice interaction is increased, the first benefit obtained by the agent is simply by obtaining some coverage of the state space with optimal advice (see the $K = 25$ line). Only after the bandwidth is increased more do we see *how* the advice is given start to make a difference.

6 Limited Bandwidth and Memory

In Section 5, the coached agent has a limited action space and there is limited bandwidth between the agents. However, the coached agent remembers all advice which the coach has sent. This is probably unrealistic once the world becomes

[1] With basic probability theory, one can calculate that if T is the number of pieces of advice and S the size of the state space, the expected number of distinct states about which the random strategy advises is $S(1 - (\frac{S-1}{S})^T)$. At the end of 700,000 steps with $K = 25$, 35000 pieces of advice have been given. With $S = 46656$, this is approximately 22035 states.

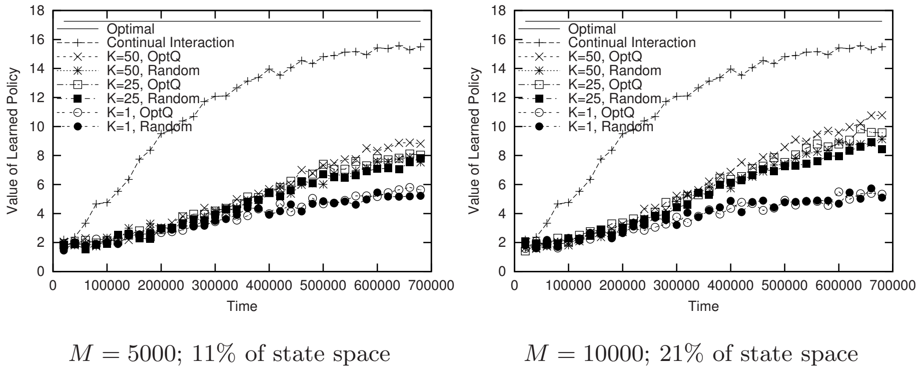

Fig. 6. Results of learning with limited predator, limited bandwidth, and limited memory, M is the number of pieces of advice the predator can remember. The $M = 1000$ case is not shown as all data lines perform at approximately the $K = 1$ level

large enough that the advised agent is not using a full state/action table. Therefore, we explore the additional limitation of varying the amount of advice which can be remembered.

6.1 Learning Algorithms

We consider a straightforward FIFO model of memory. The coached agent has a fixed memory size, and when it is full, the agent forgets the oldest advice.

The coach strategies are the same as before: RandomState (Table 4) and OptQ (Table 6). The coached agent can still uses LimitedBWAdviceTaker (Table 5), but now the state advice table T only stores the last M pieces of advice heard, where M is an independent variable which we experimentally vary.

6.2 Experimental Results in Predator-Prey

We once again use the predator prey world as described in Section 3.2. The predator has limited actions as described in Section 4.2. Figure 6 shows the results. With the smallest memory of $M = 1000$, the predator does not improve significantly over having no coach at all. This can be explained simply because the memory can only hold approximately 2% of the state space. As the amount of memory is increased, the agent's performance improves. It should be noted that for $K = 1$, throughout the entire simulation the coach only sends 1400 pieces of advice, just barely more than the smallest memory. Therefore, we can not expect increasing the memory of the to improve performance for $K = 1$ since memory is not really the limiting resource for most of the simulation.

We see the same general effects of the differences between the RandomState and OptQ strategies here as when there was no limited memory (Section 5.2). The difference is that the size of memory effectively lowers the absolute performance of all techniques. It is interesting to note that absolute performance improves (in the $M = 5000$ and $M = 10000$ cases) when moving from $K = 25$ to $K = 50$ only when using the OptQ strategy and not RandomState.

7 Conclusion

This paper examines the problem of how one automated agent (the coach) can give effective advice to another learning agent (the advisee). Our focus has been on proposing and exploring modes of interaction between the agents, as well as various limitations on the agents. We explore both problems of giving and receiving advice.

A thorough experimental analysis was done in a controlled predator-prey environment. The predator agent is a Q-learning agent that takes advice by probabilistically following the advised action. The coach agent was given a large amount of information about the world, such as the full dynamics and full information about the value and optimality of all actions.

The results consistently show that advice indeed improves the advisee's performance under all forms of limitation tested. For constant advice, our results show that the more an agent listens to coach advice, the better it performs, with the expected diminishing returns. Further when the bandwidth between coach and agent is limited, two effects are observed. First, for smaller bandwidths the presence of advice helps regardless of how that advice is chosen. Only when the bandwidth is increased further does the choice of which states to advise about begin to matter. Lastly, limiting the memory of the agent lowers overall performance, but the same general trends still hold.

Further, we have shown how the coach can learn to adapt its advice to the limitations of the agent. We show empirically that the coach is effectively learning about the actions that the limited agent can do.

The challenge of coaching involves learning about an environment, learning about the agents, and effectively communicating advice while taking into account agent abilities. We believe that this work is a core step towards completely answering that challenge.

References

1. Riley, P., Veloso, M., Kaminka, G.: An empirical study of coaching. In Asama, H., Arai, T., Fukuda, T., Hasegawa, T., eds.: Distributed Autonomous Robotic Systems 5. Springer-Verlag (2002) 215–224
2. Riley, P., Veloso, M.: Planning for distributed execution through use of probabilistic opponent models. In: AIPS-2002. (2002) 72–81 *Best Paper Award.*
3. Riley, P., Veloso, M., Kaminka, G.: Towards any-team coaching in adversarial domains. In: AAMAS-02. (2002) 1145–1146
4. Maclin, R., Shavlik, J.W.: Creating advice-taking reinforcement learners. Machine Learning **22** (1996) 251–282
5. Gordon, D., Dubramanian, D.: A multi-strategy learning scheme for knowledge assimilation in embedded agents. Informatica **17** (1993)

6. Noelle, D., Cottrell, G.: A connectionist model of instruction following. In Moore, J.D., Lehman, J.F., eds.: Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society, Hillsdale, NJ, Lawrence Erlbaum Associates (1995) 369–374
7. Clouse, J.A., Utgoff, P.E.: A teaching method for reinforcement learning. In: ICML-92. (1992) 92–101
8. Clouse, J.: Learning from an automated training agent. In Gordon, D., ed.: Working Notes of the ICML '95 Workshop on Agents that Learn from Other Agents, Tahoe City, CA (1995)
9. Sammut, C., Hurst, S., Kedzier, D., Michie, D.: Learning to fly. In: ICML-92, Morgan Kaufmann (1992)
10. Riley, P.: MPADES: Middleware for parallel agent discrete event simulation. In Kaminka, G.A., Lima, P.U., Rojas, R., eds.: RoboCup-2002: The Fifth RoboCup Competitions and Conferences. Number 2752 in Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin (2003) *RoboCup Engineering Award* (to appear).