Advice Generation from Observed Execution: Abstract Markov Decision Process Learning

Patrick Riley and **Manuela Veloso**[*]
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3891
pfr@cs.cmu.edu and mmv@cs.cmu.edu

Abstract

An advising agent, a coach, provides advice to other agents about how to act. In this paper we contribute an advice generation method using observations of agents acting in an environment. Given an abstract state definition and partially specified abstract actions, the algorithm extracts a Markov Chain, infers a Markov Decision Process, and then solves the MDP (given an arbitrary reward signal) to generate advice. We evaluate our work in a simulated robot soccer environment and experimental results show improved agent performance when using the advice generated from the MDP for both a sub-task and the full soccer game.

Introduction

A coach agent provides advice to other agent(s) to improve their performance. We focus on a coach that analyzes past performance to generate advice. The synthesis of observed executions in a manner that facilitates advice generation is a challenging problem.

Observations that do not explicitly include the actions taken by the agents are an additional challenge. The intended actions must be inferred from observed behavior. In this paper we present algorithms to learn a model, including actions, based on such observations. The model is then used to generate executable advice for agents.

The areas of advice reception (e.g. Maclin & Shavlik 1996) and advice generation, in both Intelligent Tutoring Systems (e.g. Paolucci, Suthers, & Weiner 1996) and item recommendation (e.g. Shani, Brafman, & Heckerman 2002), have received attention in AI over many years. Our work in this paper further explores advice generation in the context of agent to agent advice.

In most coaching environments, it is impractical for the coach to provide advice only at the most detailed level of states and actions because of communication bandwidth or observability constraints. Further, in domains with large or continuous state spaces, abstraction can produce the necessary generalization. Constructing an abstract model of the environment is one way that a coach may generate advice. Markov Decision Processes (MDPs) are a well-studied formalism for modeling an environment and finding a good action policy. The use of state and action abstraction has gotten significant attention (Dearden & Boutilier 1997; Barto & Mahadevan 2003; Uther & Veloso 2002). Here we introduce an approach to construct an MDP that uses both state abstraction and temporally extended abstract actions.

The MDP is learned by observation of past performance of agents in the domain, considered as a series of states that do not include information about what (abstract) actions were performed. This type of data can be obtained by an external observer with no access to the internal processes of the agents. Domain knowledge is used to transform states to abstract states and then to attach actions to transitions, whose probabilities are estimated from observed data. This process of adding actions has much in common with past work in plan recognition from observation (Kautz 1991; Charniak & Goldman 1993).

We used a simulated robot soccer environment as the testbed. The presence of a coach agent makes the environment well suited to this research. Past work on coaching for this environment has mostly been on analysis of a particular team, with automated (Kuhlmann, Stone, & Lallinger 2004; Riley, Veloso, & Kaminka 2002; Visser & Weland 2004) or non-automated (Raines, Tambe, & Marsella 2000) advice generation. We present experimental results for a coach agent learning an MDP and improving performance for both a restricted part of the game and in full games.

Learning an Abstract MDP from Observation

In this section we introduce our approach for creating a Markov Decision Process from observations. Our goal is to learn an *abstract* MDP. The main steps of the process are:

- Transform observations into sequences of abstract states.
- Create a Markov Chain based on the observed transition frequencies among the abstract states.
- Transform the Markov Chain to a Markov Decision Process (MDP) by associating transitions to actions based on a specification of the abstract actions to add.
- Add rewards to the MDP.

[*] This research was sponsored by Grants No. F30602-00-2-0549 and the Department of the Interior (DOI) - National Business Center (NBC) and the Defense Advanced Research Projects Agency (DARPA) under contract no. NBCHC030029. The content of this publication reflects only the position of the authors.

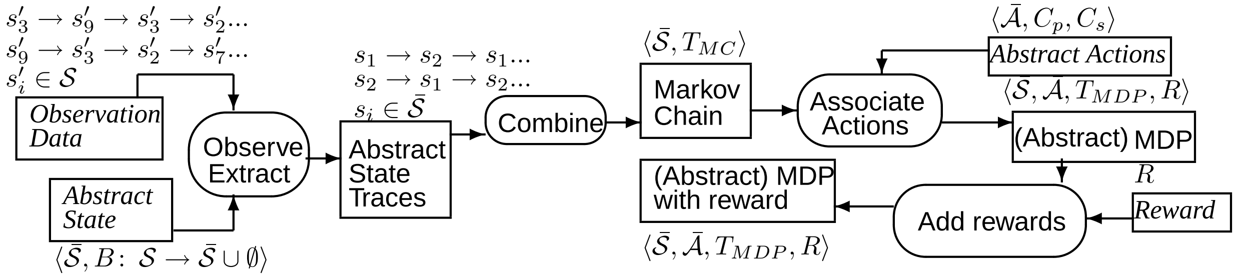

$s'_3 \to s'_9 \to s'_3 \to s'_2 ...$
$s'_9 \to s'_3 \to s'_2 \to s'_7 ...$
$s'_i \in \mathcal{S}$
Observation Data
Abstract State
$\langle \bar{\mathcal{S}}, B\colon \mathcal{S} \to \bar{\mathcal{S}} \cup \emptyset \rangle$
Observe Extract
$s_1 \to s_2 \to s_1 ...$
$s_2 \to s_1 \to s_2 ...$
$s_i \in \bar{\mathcal{S}}$
Abstract State Traces
Combine
$\langle \bar{\mathcal{S}}, T_{MC} \rangle$
Markov Chain
Associate Actions
$\langle \bar{\mathcal{A}}, C_p, C_s \rangle$
Abstract Actions
$\langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, T_{MDP}, R \rangle$
(Abstract) MDP
R
Reward
Add rewards
(Abstract) MDP with reward
$\langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, T_{MDP}, R \rangle$

Figure 1: The process of a coach learning a Markov Decision Process from observation. The symbols are examples or types based on the formalism presented. The boxes with italics indicate information that must be provided for the system.

Figure 1 depicts the processes and data involved and we now describe the algorithms in detail.

Observations to Markov Chain

There are two initial inputs to the process:

Observation data consisting of sequences of observed states. These sequences can come from recordings of past performance or from online observation. Let $\mathcal{S}$ be the observed state space. The observation data is then a list of sequences of $\mathcal{S}$, or in other words, a list of $\mathcal{S}^*$.[1]

State abstraction consisting of an abstract state space $\bar{\mathcal{S}}$, and an abstraction function $B\colon \mathcal{S} \to \bar{\mathcal{S}} \cup \emptyset$. Note that B can map elements of $\mathcal{S}$ to the empty set, indicating there is no corresponding abstract state. Naturally, this specification can have a large impact on the overall performance on the system. While we currently use our domain expertise to specify the abstraction, learning techniques do exist which may provide useful abstractions (e.g. Schweitzer, Puterman, & Kindle 1985).

Observe and extract as shown in Figure 1 can then be implemented in terms of the above. B is applied to every element of every sequence in the observation data. Any elements that map to $\emptyset$ are removed. *Observe and extract* outputs *abstract state traces* as a list of $\bar{\mathcal{S}}^*$.

Given the state traces, the *combine* algorithm produces a Markov Chain, a tuple $\langle \bar{\mathcal{S}}, T_{MC} \rangle$ where $\bar{\mathcal{S}}$ is the set of abstract states and $T_{MC}\colon \bar{\mathcal{S}} \times \bar{\mathcal{S}} \to \mathbb{R}$ is the transition function. $T_{MC}(s_1, s_2)$ gives the probability of transitioning from s_1 to s_2. Since the state space $\bar{\mathcal{S}}$ for the Markov Chain has already been given, just T_{MC} must be calculated.

The *combine* algorithm estimates the transition probabilities based on the observed transitions. The algorithm calculates, for every pair of states $s_1, s_2 \in \bar{\mathcal{S}}$, a value c_{s_1,s_2} which is the number of times a transition from s_1 to s_2 was observed. The transition function is then, $\forall s_1, s_2 \in \bar{\mathcal{S}}$:

$$T_{MC}(s_1, s_2) = \frac{c_{s_1,s_2}}{\sum_{s \in \bar{\mathcal{S}}} c_{s_1,s}} \tag{1}$$

This estimation of T_{MC} could possibly benefit from smoothing or clustering techniques to deal with problems of data sparsity. Our use of state abstraction already helps to deal with this somewhat and our experimental results demonstrate usefulness of the resulting model. More sophisticated estimation is an avenue for future work.

Markov Chain to Markov Decision Process

Our algorithm now converts the Markov Chain into a Markov Decision Process. A Markov Decision Process is a tuple $\langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, T_{MDP}, R \rangle$. $\bar{\mathcal{S}}$ is the set of abstract states, $\bar{\mathcal{A}}$ is the set of (abstract) actions, $T_{MDP}\colon \bar{\mathcal{S}} \times \bar{\mathcal{A}} \times \bar{\mathcal{S}} \to \mathbb{R}$ is the transition function, and $R\colon \bar{\mathcal{S}} \to \mathbb{R}$ is the reward function. Similar to a Markov Chain, the transition function $T_{MDP}(s_1, a, s_2)$ gives the probability of transitioning from s_1 to s_2 given that action a was taken.

Shani, Brafman, & Heckerman (2002) also explain a mechanism for Markov Chain to Markov Decision Process conversion. However, for them the Markov Chain describes how the system acts with no agent actions. They use a simple proportional probability transform to add the actions. Here, the Markov Chain represents the environment with agent actions and we need to infer what those actions are.

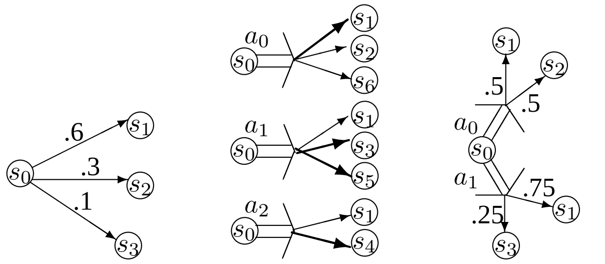
In our approach, a set of abstract actions must be given as a set of transitions between abstract states, divided into primary and secondary transitions. The primary transitions represent the intended or normal effects of actions and the secondary transitions are other possible results of the action. This distinction is similar to Jensen, Veloso, & Bryant (2004).

We initially assign a zero reward function. Describing the algorithms for associating actions and transitions takes up the bulk of this section. We first illustrate the process using the example shown in Figure 2, then provide the full details.

Our algorithm processes each abstract state in turn. Figure 2(a) shows the state s_0 in the Markov Chain. It must be determined which actions can be applied in each state and how to assign the transitions and their probabilities to actions.

The transitions for the actions a_0, a_1, and a_2 are shown in Figure 2(b), with the primary transitions in bold. An action is added for a state if any of the action's primary transitions exist for this state. In Figure 2(c), actions a_0 and a_1 have been added, but a_2 has not (since its primary transition $s_0 \to s_4$ was not in the Markov Chain). Once an action has

[1] The following notation will be used in the remainder of the paper. Sets will be set in script, e.g. $\mathcal{S}, \mathcal{N}$. The notation $\mathcal{S}^i$ is used to denote a sequence of i elements of $\mathcal{S}$. $\mathcal{S}^*$ will denote $\cup_{i \in \mathbb{N}} \mathcal{S}^i$. The powerset of a set $\mathcal{S}$ will be denoted by $\mathcal{P}(\mathcal{S})$. Functions will be capital letters and in italics, e.g. F, C.



(a) Markov Chain (b) Actions (c) MDP

Figure 2: Associating actions with the transitions from one state in a Markov Chain to create a Markov Decision Process state

been added, all transitions in the Markov Chain which are primary or secondary transitions for the action are assigned to the action. It is fine for primary or secondary transitions that are part of the action definition to not be in the Markov Chain (e.g. the $s_0 \rightarrow s_6$ transition for a_0 and the $s_0 \rightarrow s_5$ for a_1). Once all actions have been processed, the probability mass for each transition is divided equally among all repetitions of the transition and the resulting distributions are normalized. For example, in Figure 2, the probability mass of 0.6 assigned to the $s_0 \rightarrow s_1$ transition is divided by 2 for the 2 repetitions. The transitions $s_0 \rightarrow s_1$ and $s_0 \rightarrow s_3$ have, respectively, probabilities 0.3 and 0.1 before normalization and 0.75 and 0.25 after normalization.

Formally, an abstract action set $\bar{\mathcal{A}}$ and functions giving the primary ($C_p\colon \bar{\mathcal{A}} \rightarrow \mathcal{P}(\bar{\mathcal{S}} \times \bar{\mathcal{S}})$) and secondary ($C_s\colon \bar{\mathcal{A}} \rightarrow \mathcal{P}(\bar{\mathcal{S}} \times \bar{\mathcal{S}})$) transitions must be given. We will calculate an unnormalized transition function T'_{MDP} which will be normalized to T_{MDP}. The complete algorithm for adding actions to the Markov Chain is shown in Table 1.

```
For all $s \in \bar{\mathcal{S}}$
  Let $\mathcal{T} \subseteq \mathcal{P}(\bar{\mathcal{S}} \times \bar{\mathcal{S}})$ be the transitions from $s$
  $\mathcal{T} = \left\{\langle s, s_i\rangle \mid s_i \in \bar{\mathcal{S}}, T_{MC}(s, s_i) > 0\right\}$
  Let $\mathcal{N} \subseteq \bar{\mathcal{A}}$ be actions with primary transitions for $s$
  $\mathcal{N} = \left\{a \in \bar{\mathcal{A}} \mid C_p(a) \cap \mathcal{T} \neq \emptyset\right\}$
  For all $s_i \in \bar{\mathcal{S}}$
      Let $c_{s_i}$ be the number of actions for $s \rightarrow s_i$
      $c_{s_i} = |\{a \in \mathcal{N} \mid \langle s, s_i\rangle \in C_p(a) \cup C_s(a)\}|$
  For all $a \in \mathcal{N}$
      For all $\langle s, s_i\rangle \in C_p(a) \cup C_s(a)$
          $T'_{MDP}(s, a, s_i) = \dfrac{T_{MC}(s, s_i)}{c_{s_i}}$
  Add null action if needed (see text)
Normalize $T'_{MDP}$ to $T_{MDP}$
```

Table 1: Associating actions to convert a Markov Chain to an MDP.

As noted, a null action can be added. This occurs if a transition from the Markov chain is not part of any of the actions created for this state (i.e. if $c_{s_i} = 0$ and $T_{MC}(s, s_i) \neq 0$).

The definition of primary and secondary transitions and the algorithm above support situations in which the possible effects of an abstract action are known, but the probabilities of occurrence of the various results are not.

Once we have a Markov Decision Process, we can add additional rewards to whatever states desired. By changing the reward function, the learned Markov process can be used to produce different behaviors. This is explored further the in empirical section below.

Learning In Simulated Robot Soccer

We used the Soccer Server System (Noda *et al.* 1998) as used in RoboCup (Kitano *et al.* 1997) as our implementation and testing environment. The Soccer Server System is a server-client system that simulates soccer between distributed agents. Clients communicate using a standard network protocol with well-defined actions. The server keeps track of the current state of the world, executes the actions the clients request, and periodically sends perceptions to the agents. Agents receive noisy information about the direction and distance of objects on the field (the ball, players, goals, etc.); information is provided only for objects in the field of vision of the agent. The agents communicate with the server at the level of actions like turn, dash, and kick. Higher level actions like passing or positioning on the field are implemented as combinations of these lower level actions.

There are eleven independent players on each side, as well as a coach agent who has a global view of the world, but whose only action is to send advice to the players.

This environment has a standard advice language called CLang, which is primarily rule based. Conditions are logical connections of domain specific atoms like ball and player location. Actions include the abstract soccer actions like passing and dribbling.

For the simulated soccer environment, only those states where an agent can kick the ball are represented in the abstract state space. We define the abstract state space in terms of a set of factors and values:

GoalScore 0,1,null, if we, they, or no one scored.

DeadBall 0,1,null, if it is our, their, or no one's free kick.

BallGrid Location of the ball on a discretized grid for the field (see Figure 3).

BallOwner 0,1,2, if no one, our, or their team owns the ball.

PlayerOccupancy Presence of teammate and opponent players in defined regions.

The regions for PlayerOccupancy for the opponents are shown in Figure 4. The regions are centered around the ball and oriented so that the right is always towards the attacking goal. The regions for our team are the same except that players which can currently kick the ball are ignored.

The abstract action space $\bar{\mathcal{A}}$ was constrained by the advice actions in CLang. CLang supports abstract actions like passing, dribbling, and clearing (kicking the ball away). All these actions can take a parameter of an arbitrary region of the field. Actions should correspond to changes in the abstract state space. Since many CLang actions involve ball movement, we chose to consider the ball movement actions with regions from the discrete grid of ball locations (see Figure 3) as the parameters.

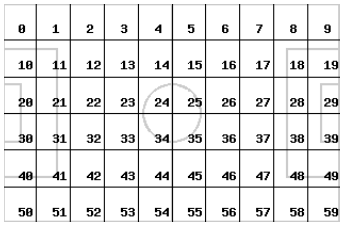


Figure 3: The regions for the BallGrid state factor

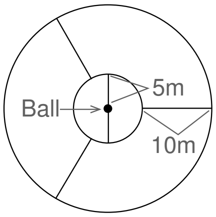


Figure 4: Player occupancy regions

To construct the C_p and C_s functions describing the primary and secondary transitions for the actions, we first classify the transitions (using the structured state representation discussed above). Then, these functions can be described in terms of the transition classes they represent, rather than writing out individual transitions. For example, we had transition classes including successful shots, kicks out of bounds, short passes, and failed passes.

Markov Decision Process to Advice

The final set of algorithms go from a Markov Decision Process to advice. The first step is to solve the MDP. Since we have all transition probabilities, we use a dynamic programming approach (Puterman 1994). This algorithm gives us a Q table, where for $s \in \bar{\mathcal{S}}$ and $a \in \bar{\mathcal{A}}$, $Q(s, a)$ gives the expected future discounted reward of taking action a in state s and performing optimally afterwords. An optimal policy (a mapping from $\bar{\mathcal{S}}$ to $\bar{\mathcal{A}}$) can be extracted from the Q table by taking the action with the highest Q value for a given state.

States and actions for advice must then be chosen. Advising about all states can overload the communication between the coach and agents (which is limited) and stress the computational resources of the agent applying the advice. Therefore, the scope of advice is restricted as follows:

- Remove states which don't have a minimum number of actions. For states with many possible actions, the agents will in general be in more need of advice from the coach. We experimented with different values here, but most of the time we only removed states without any actions.
- Remove states whose optimal actions can't be translated into the advice language.
- Only advise actions which are close to optimal. We only want to advise "good" actions, but we must be specific about what good means. We only advise actions which are within a given percentage of optimal for the given state.

Once the states and actions which to advise have been determined, they must be translated into the advice language.

Advising In Simulated Robot Soccer

For simulated robot soccer, we pruned any action dealing with actions that the opponent takes. Clearly, we can not advise the agents to perform these actions.

After pruning, we are left with a set of pairs of abstract states and abstract actions. The advice is translated in to the CLang advice language. The goal is to structure the advice such that it can be matched and applied quickly at run time by the agents. We use the structured abstract state representation discussed above to construct a tree of CLang rules. At each internal node, one factor from the state representation is matched. At each leaf, a set of actions (for the abstract state matched along the path to the leaf) is advised.

Empirical Validation

The learning and advice generation system is fully implemented for the SoccerServer (Noda *et al.* 1998). A version of the system described here made up the bulk of the Owl entry to the RoboCup 2003 coach competition. This section describes our empirical work in validating the system.

Throughout, we have used two different teams of agents which understand the advice language CLang: UTAustin Villa from the University of Texas at Austin (UTA) and the Wyverns from Carnegie Mellon University (CM).[2]

The data files, binaries, and configuration for all of these experiments can be found at the paper's online appendix: `http://www.cs.cmu.edu/~pfr/appendices/2004aaai.html`.

Circle Passing

We constructed a sub-game of soccer in order to clearly evaluate the MDP learning from observed execution and the effect of automatically generated advice. While we are interested in modeling the entire soccer game, the number of factors affecting performance make it difficult to separately test effects.

We set up agents in a circle around the middle of the field and wrote a set of advice rules which cause the agents to pass in a circle, as shown in Figure 5. Stamina and offsides were turned off and dead balls were put back into play after 1 second. The goal was to use this data to learn a model which then be used to generate advice for a different team. This team would not know of the performance of the original team or even what the reward states in the model are. The goal was not to replicate this passing pattern, but to achieve the reward specified.

We ran 90 games of the UTA players executing this passing pattern. Note that because of noise in the environment, not every pass happens as intended. Agents sometimes fail to receive a pass and have to chase it outside of their normal positions. Kicks can be missed such that the ball goes in a direction other than intended or such that it looks like the agent is dribbling before it passes. These "errors" are important for the coach; it allows the coach observe other possible actions and results of the original advice.

We ran the process described above on the 90 games of data. Since there were no opponents on the field, the total possible size of the abstract state space was 5882 states. The effective state space size (states actually observed) was 346 states. Our algorithm produced a Markov Chain and a Markov Decision Process from this data.

[2] While both these institutions contributed agents for the official RoboCup2003 coachable team, the versions of the agents used here are updated from those official releases.

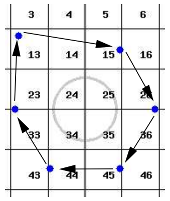


Figure 5: Locations and directions of passing for circle passing (cf Figure 3).

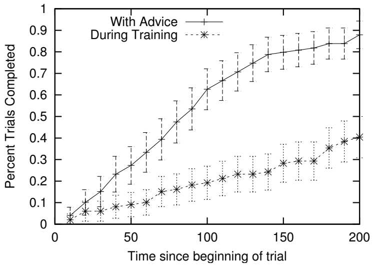


Figure 6: Time taken for trials in the reward cell 34 scenario. The intervals are 95% confidence intervals.

We experimented with adding different reward functions. In each case, reward was associated with all states in which the ball was in a particular grid cell (see Figure 5), namely:

Cell 13 The cell where the upper left player usually stands.

Cell 3 The cell immediately above the previous one. No agent would normally be here, but some passes and miskicks will result in agents being in this square.

Cell 34 Near the middle of the field. Agents tend to move towards the ball (especially to try and receive a pass) so agents often end up in this cell during training.

Cell 14 To the right of the upper left player. Since this is in the normal path between two players passing, the ball will frequently pass through this square. On a miskick, either one of the two closest agents could end up in that square with the ball.

There are other reasonable reward functions. We chose these to vary the degree to which the reward states were observed during training. Similar states around any one of the players would likely give similar results.

We ran with the CM agents receiving advice in each of these scenarios. Actions were sent if they were within 99.9% of optimal. We randomly chose 100 spots in the area around the players and for each of the eight cases (4 different rewards and training or with MDP advice) put the ball in each of those spots. The agents then ran for 200 cycles (20 seconds). A trial was considered a success if the agents got to any reward state in that time. The time bound is somewhat arbitrary as varying the time bound somewhat does not significantly affect the relative results. Further, the completion results at a particular time are easier to present and discuss that the full series of rewards received. Table 2 shows the results for the agents executing the learned advice for the four different reward cells. The "Training Success" line shows the percent of trials which completed during the initial UTA training games as a basis for comparison.

In all scenarios, the success percentage is higher with the MDP based advice. Figure 6 shows a different view of the reward cell 34 scenario. The x-axis is the time since the beginning of a trial and the y-axis is the percentage of the trials which received reward by that time. Graphs for the other scenarios are similar.

Reward Grid Cell	13	3	34	14
# rew. states observed	5095	211	1912	2078
Training Success %	53%	4%	40%	44%
Advice Success %	77%	21%	88%	69%

Table 2: Performance for circle passing. The first row shows the number of times a reward state was seen during training.

In all cases we see that agent execution is not perfect. This occurs for several reasons. Noise in the perception and execution can cause actions to fail and have undesired effects. Execution of some abstract actions (such as passing) requires more about the underlying states than what is expressed by the abstraction. In passing, another agent needs to be in or near the target location in order to receive the pass. Therefore, it can happen that the advised abstract action can not be done at this time.

The reward cell 3 scenario was the most difficult for the agents to achieve. It was also the set of reward states which was least often seen in training. More examples of transitions to reward states allow a better model to be created.

These experiments demonstrate that the coach can learn a model from observation which can then be used to improve agent performance. Changing the reward can allow the same transition model to be used to generate different advice. However, the more similar the training data is to the desired execution, the more effective the advice is in helping the agents achieve reward.

Full Game

While the above experiments show that the MDP learning process described can extract a useful model and generate advice from it, providing advice for an entire soccer game is a larger and more challenging task. This section demonstrates that our coach can produce positive effects on a team's overall performance. While further experiments exploring the limits and general effectiveness would be useful, these results are still a compelling example of improvement.

Another advantage of the simulated robot soccer environment can also be leveraged here. Annual worldwide competitions have been held since 1997 with a number of smaller regional competitions over the last few years. Most of the logfiles from these competitions have been preserved and are publicly available. This is a wealth of data of many different teams playing. We used all the logfiles from RoboCup2001 and 2002, German Open 2002 and 2003, Japan Open 2002 and 2003, American Open 2003, and Australian Open 2003. This is a total of 601 logfiles. We also analyzed 1724 logfiles from our previous experiments with a number of past teams (Riley, Veloso, & Kaminka 2002).

While it may be better to only analyze games played between these particular teams or at least just involving this particular opponent, we wanted to take advantage of the wealth of past games available for analysis. Observing a range of performance should hopefully allow an understanding of the average case. While some of the knowledge learned could be internally inconsistent, we hope that most such cases will average out over time. Use of additional observations of the particular teams in order to refine the model is another avenue for further exploration.

Given the definition of the abstract state space above, there are 184442 possible states. After analysis, the MDP contained 89223 states. We associated a reward of 100 with our team scoring a goal and -100 for being scored upon. We also removed all transitions from these states. This allows for faster convergence of the dynamic programming because rewards do not have to be backed up through these states.

We tested two conditions: the CM team playing against Sirim (one of the fixed opponents in the RoboCup2003 coach competition) with and without the MDP based advice. With the MDP advice, actions were included if they were at least 96% of optimal. Each condition was run for 30 games. The results in Table 3 show that using the MDP improves the score difference by an average of 1.6 goals. This difference is significant at a $< 1\%$ level for a one-tailed t-test.

	No MDP	With MDP
Score Difference	-4.6 [-5.3, -3.8]	-3 [-3.5, -2.5]

Table 3: Mean score difference of CM playing Sirim. Score difference is CM's score minus Sirim's score. The interval shown is the 95% confidence interval.

While these results do not demonstrate the coached team moving from losing to winning, the results still show a significant improvement in performance. The overall performance of a simulated soccer team is a combination of many factors, including low level system timing and synchronization, implementation of basic skills like kicking and dribbling, and higher level strategy decisions. Coaching advice can only affect the last of these. As far as we are aware, our coach is the first for the simulated robot soccer environment that advises about such a large portion of the behaviors of the agents and does so in an entirely learned fashion.

Conclusion

This paper has examined the problem of learning a model of an environment in order to generate advice. An MDP is learned based on observations of past agent performance in the environment and domain knowledge about the structure of abstract states and actions in the domain. The MDP learning process first creates a Markov Chain. Domain knowledge about actions is then used to successfully transform the Markov Chain into an MDP. Implementation was done in a simulated robot soccer environment. In two different scenarios, the advice generated from the learned MDP was shown to improve the performance of the agents.

This research provides a crucial step in agent to agent advice giving, namely the automatic generation of effective, executable advice from raw observations.

References

Barto, A., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete-Event Systems Journal* 13:41–77.

Charniak, E., and Goldman, R. 1993. A Bayesian model of plan recognition. *Artificial Intelligence* 64(1):53–79.

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision theoretic planning. *Artificial Intelligence* 89(1):219–283.

Jensen, R. M.; Veloso, M. M.; and Bryant, R. E. 2004. Fault Tolerant Planning: Toward Probabilistic Uncertainty Models in Symbolic Non-Deterministic Planning. In *ICAPS04*.

Kautz, H. A. 1991. A Formal theory of plan recognition and its implementation. In Allen, J. F.; Kautz, H. A.; Pelavin, R. N.; and Tenenberg, J. D., eds., *Reasoning About Plans*. Los Altos, CA: Morgan Kaufmann. chapter 2.

Kitano, H.; Tambe, M.; Stone, P.; Veloso, M.; Coradeschi, S.; Osawa, E.; Matsubara, H.; Noda, I.; and Asada, M. 1997. The RoboCup synthetic agent challenge. In *IJCAI-97*, 24–49.

Kuhlmann, G.; Stone, P.; and Lallinger, J. 2004. The champion UT Austin Villa 2003 simulator online coach team. In Polani, D.; Browning, B.; Bonarini, A.; and Yoshida, K., eds., *RoboCup-2003: Robot Soccer World Cup VII*. Berlin: Springer Verlag. (to appear).

Maclin, R., and Shavlik, J. W. 1996. Creating advice-taking reinforcement learners. *Machine Learning* 22:251–282.

Noda, I.; Matsubara, H.; Hiraki, K.; and Frank, I. 1998. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence* 12(2–3):233–250.

Paolucci, M.; Suthers, D. D.; and Weiner, A. 1996. Automated advice-giving strategies for scientific inquiry. In *ITS-96*, 372–381.

Puterman, M. L. 1994. *Markov Decision Processes*. New York: John Wiley & Sons.

Raines, T.; Tambe, M.; and Marsella, S. 2000. Automated assistant to aid humans in understanding team behaviors. In *Agents-2000*.

Riley, P.; Veloso, M.; and Kaminka, G. 2002. An empirical study of coaching. In Asama, H.; Arai, T.; Fukuda, T.; and Hasegawa, T., eds., *Distributed Autonomous Robotic Systems 5*. Springer-Verlag. 215–224.

Schweitzer, P. L.; Puterman, M. L.; and Kindle, K. W. 1985. Iterative aggregation-deaggregation procedures for discounted semi-Markov reward processes. *Operations Research* 33:589–605.

Shani, G.; Brafman, R. I.; and Heckerman, D. 2002. An MDP-based recommender system. In *UAI-2002*, 453–460.

Uther, W., and Veloso, M. 2002. TTree: Tree-based state generalization with temporally abstract actions. In *Proceedings of SARA-2002*.

Visser, U., and Weland, H.-G. 2004. Using online learning to analyze the opponent behavior. In Polani, D.; Bonarini, A.; Browning, B.; and Yoshida, K., eds., *RoboCup-2003: The Sixth RoboCup Competitions and Conferences*. Berlin: Springer Verlag. (to appear).