In *The Eighteenth International Conference on Machine Learning* (ICML 2001)
pp. 537-544, Williamstown, MA, USA, June 2001.

Scaling Reinforcement Learning toward RoboCup Soccer

Peter Stone PSTONE@RESEARCH.ATT.COM
Richard S. Sutton SUTTON@RESEARCH.ATT.COM
AT&T Shannon Laboratory, 180 Park Ave., Florham Park, NJ 07932 USA

Abstract

RoboCup simulated soccer presents many challenges to reinforcement learning methods, including a large state space, hidden and uncertain state, multiple agents, and long and variable delays in the effects of actions. We describe our application of episodic SMDP Sarsa(λ) with linear tile-coding function approximation and variable λ to learning higher-level decisions in a keepaway subtask of RoboCup soccer. In keepaway, one team, "the keepers," tries to keep control of the ball for as long as possible despite the efforts of "the takers." The keepers learn individually when to hold the ball and when to pass to a teammate, while the takers learn when to charge the ball-holder and when to cover possible passing lanes. Our agents learned policies that significantly out-performed a range of benchmark policies. We demonstrate the generality of our approach by applying it to a number of task variations including different field sizes and different numbers of players on each team.

1. Introduction

RoboCup simulated soccer has been used as the basis for successful international competitions and research challenges (Kitano et al., 1997). As presented in detail by Stone (2000), it is a fully *distributed*, *multiagent* domain with both *teammates* and *adversaries*. There is *hidden state*, meaning that each agent has only a partial world view at any given moment. The agents also have *noisy sensors and actuators*, meaning that they do not perceive the world exactly as it is, nor can they affect the world exactly as intended. In addition, the perception and action cycles are *asynchronous*, prohibiting the traditional AI paradigm of using perceptual input to trigger actions. *Communication* opportunities are limited, and the agents must make their decisions in *real-time*. These italicized domain characteristics combine to make simulated robotic soccer a realistic and challenging domain.

In principle, modern reinforcement learning methods are reasonably well suited to meeting the challenges of RoboCup simulated soccer. Reinforcement learning is all about sequential decision making, achieving delayed goals, and handling noise and stochasticity. It is also oriented toward making decisions relatively rapidly rather than relying on extensive deliberation or meta-reasoning. There is a substantial body of reinforcement learning research on multiagent decision making, and soccer is an example of the relatively benign case in which all agents on the same team share the same goal. In this case it is often feasible for each agent to learn independently, sharing only a common reward signal. The problem of hidden state remains severe, but can, again in principle, be handled using function approximation, which we discuss further below. RoboCup soccer is a large and difficult instance of many of the issues which have been addressed in small, isolated cases in previous reinforcement learning research. Despite substantial previous work (e.g., Andou, 1998; Stone & Veloso, 1999; Uchibe, 1999; Riedmiller et al., 2001), the extent to which modern reinforcement learning methods can meet these challenges remains an open question.

Perhaps the most pressing challenge in RoboCup simulated soccer is the large state space, which requires some kind of general function approximation. Stone and Veloso (1999) and others have applied state aggregation approaches, but these are not well suited to learning complex functions. In addition, the theory of reinforcement learning with function approximation is not yet well understood (e.g., see Sutton & Barto, 1998; Baird & Moore, 1999; Sutton et al., 1999). Perhaps the best understood of current methods is linear Sarsa(λ), which we use here. This method is known not to converge in the conventional sense, but several lines of evidence suggest that it nevertheless remains near a good solution (Gordon, 2001; Tsitsiklis & Van Roy, 1997; Sutton, 1996). Certainly it has advantages over off-policy methods such as Q-learning, which can be unstable with linear and other kinds of function approximation. An important open question is whether Sarsa's failure to converge in the conventional sense is of practical importance or is merely a theoretical curiosity. Only tests on large-state-space applications such as RoboCup soccer will answer this question.

In this paper we begin to scale reinforcement learning up to RoboCup simulated soccer. We consider a subtask of soccer involving 5–7 players rather than the full 22. This is the task of *keepaway*, in which one team merely seeks to keep control of the ball as long as possible. In the next section we describe keepaway and how we build on prior work in RoboCup soccer to formulate this problem at an intermediate level above that of the lowest level actions and perceptions. In Section 3 we map this task onto an episodic reinforcement learning framework, and in Sections 4 and 5 we describe our learning algorithm in detail and our results respectively. Related work is discussed further in Section 6.

2. Keepaway Soccer

We consider a subtask of RoboCup soccer, *keepaway*, in which one team, the *keepers*, is trying to maintain possession of the ball within a limited region, while another team, the *takers*, is trying to gain possession. Whenever the takers take possession or the ball leaves the region, the *episode* ends and the players are reset for another episode (with the keepers being given possession of the ball again).

Parameters of the task include the size of the region, the number of keepers, and the number of takers. Figure 1 shows a screen shot of 3 keepers and 2 takers (called 3 vs. 2, or 3v2 for short) playing in a 20m x 20m region.

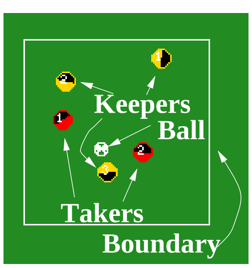

Figure 1: A screen shot from a 3 vs. 2 keepaway episode in a 20m x 20m region.

All of our work uses version 5.25 of the standard RoboCup soccer simulator (Noda et al., 1998). An omniscient coach agent manages the play, ending episodes when a taker gains possession of the ball for a set period of time or when the ball goes outside of the region. At the beginning of each episode, the coach resets the location of the ball and of the players semi-randomly within the region of play. The takers all start in one corner (bottom left). Three randomly chosen keepers are placed one in each of the three remaining corners, and any keepers beyond three are placed in the center of the region. The ball is initially placed next to the keeper in the top left corner.

In the RoboCup soccer simulator, agents typically have limited and noisy sensors: each player can see objects within a 90^o view cone, and the precision of an object's sensed location degrades with distance. To simplify the problem, we gave our agents (both keepers and takers) a full 360^o of noise-free vision (object movements and agent actions retain their usual noisy characteristics). We have not yet determined whether this simplification was necessary or helpful[1].

Keepaway is a subproblem of robotic soccer. The principal simplifications are that there are fewer players involved; they are playing in a smaller area; and the players are always focused on the same high-level goal—they don't need to balance offensive and defensive considerations. Nevertheless, the skills needed to play keepaway well are also very useful in the full problem of robotic soccer. Indeed, ATT-CMUnited-2000—the 3rd-place finishing team in the RoboCup-2000 simulator league—incorporated a successful hand-coded solution to an 11 vs. 11 keepaway task (Stone & McAllester, 2001).

3. Mapping Keepaway onto Reinforcement Learning

Our keepaway problem maps fairly directly onto the discrete-time, episodic, reinforcement-learning framework. The RoboCup soccer simulator operates in discrete time steps, $t = 0, 1, 2, \ldots$, each representing 100 *msec* of simulated time. When one episode ends (e.g., the ball is lost to the takers), another begins, giving rise to a series of episodes. Each player learns independently and may perceive the world differently. For each player, an episode begins when the player is first asked to make a decision and ends when possession of the ball is lost by the keepers.

As a way of incorporating domain knowledge, our learners chose not from the simulator's primitive actions, but from higher level actions constructed from skills used in the CMUnited-99 team. Those skills included

[1] Preliminary tests indicate that at least the noise-free vision is not essential.

HoldBall(): Remain stationary while keeping possession of the ball in a position that is as far away from the opponents as possible.

PassBall(k): Kick the ball directly towards keeper k.

GetOpen(): Move to a position that is free from opponents and open for a pass from the ball's current position (using SPAR (Veloso et al., 1999)).

GoToBall(): Intercept a moving ball or move directly towards a stationary ball.

BlockPass(k): Move to a position between the keeper with the ball and keeper k.

All of these skills except PassBall(k) are simple functions from state to a corresponding action; an invocation of one of these normally controls behavior for a single time step. PassBall(k), however, requires an extended sequence of actions—getting the ball into position for kicking, and then a sequence of kicks in the desired direction (Stone, 2000); a single invocation of PassBall(k) influences behavior for several time steps. Moreover, even the simpler skills may last more than one time step because the player occasionally misses the step following them; the simulator occasionally misses commands; or the player may find itself in a situation requiring it to take a specific action, for instance to self-localize. In these cases there is no new opportunity for decision making until two or more steps after invoking the skill. To handle such possibilities, it is convenient to treat the problem as a *semi-Markov* decision process, or SMDP (Puterman, 1994; Bradtke & Duff, 1995). An SMDP evolves in a sequence of jumps from the initiation of each SMDP action to its termination one or more time steps later, at which time the next SMDP action is initiated. SMDP actions that consist of a subpolicy and termination condition over an underlying decision process, as here, have been termed *options* (Sutton et al., 1999).

Let $t_0, t_1, t_2, \ldots, t_j$ denote the SMDP time steps, those at which options are initiated and terminated, where t_0 denotes the first time step of the episode and t_j the last (all options terminate at the end of an episode). From the point of view of the SMDP, then, the episode consists of a sequence of states, options, and rewards

$$s_0, a_0, r_1, s_1, \ldots, s_i, a_i, r_{i+1}, s_{i+1}, \ldots, r_j, s_j$$

where s_i denotes the state of the simulator at time step t_i, a_i denotes the option initiated at t_i based on some, presumably incomplete, perception of s_i, and $r_{i+1} \in \Re$ and s_{i+1} represent the resultant reward and state at the termination of a_i. The final state of the episode, s_j is the state where the takers have possession or the ball has gone out of bounds. We wish to reward the keepers for each time step in which they keep possession, so we set $r_i = t_i - t_{i-1}$. The takers, on the other hand, are penalized for each time step, so their reward is just the inverse: $t_{i-1} - t_i$. Because the task is episodic, no discounting is necessary: the goal at each learning step is to take an action such that the remainder of the episode will be as long or as short (keeper or taker) as possible, and thus to maximize total reward.

3.1 Keepers

Our experiments investigated learning by the keepers when in possession[2] of the ball. Keepers not in possession of the ball are required to select the Receive option:

Receive: If a teammate possesses the ball, or can get to the ball faster than this keeper can, invoke GetOpen() for one step; otherwise, invoke GoToBall() for one step. Repeat until a keeper has possession of the ball or the episode ends.

A keeper in possession, on the other hand, is faced with a genuine choice. It may hold the ball, or it may pass to one of its teammates. That is, it chooses an option from {Holdball, PassK_2ThenReceive, PassK_3ThenReceive, ..., PassK_nThenReceive} where the Holdball option simply executes HoldBall() for one step (or more if, for example, the server misses the next step) and the Pass$k$ThenReceive options involve passes to the other keepers. The keepers are numbered by their closeness to the keeper with the ball: K_1 is the keeper with the ball, K_2 is the closest keeper to it, K_3 the next closest, and so on up to K_n, where n is the number of keepers. Each Pass$k$ThenReceive is defined as

Pass$k$ThenReceive: Invoke PassBall(k) to kick the ball toward teammate k. Then behave and terminate as in the Receive option.

The keepers' learning process thus searches a constrained policy space characterized only by the choice of option when in possession of the ball. Examples of policies within this space are provided by our benchmark policies:

Random: Choose randomly among the n options, each with probability $\frac{1}{n}$.

Hold: Always choose HoldBall()

Hand-coded:

If no taker is within 10m, choose HoldBall();

Else If teammate k is in a better location than the keeper with the ball and the other team-

[2] "Possession" in the soccer simulator is not well-defined because the ball never occupies the same location as a player. One of our agents considers that it has possession of the ball if the ball is close enough to kick it.

mates, and the pass is likely to succeed (using the CMUnited-99 pass-evaluation function, which is trained off-line using the C4.5 decision tree training algorithm (Quinlan, 1993)), then choose PassBall(k);

Else choose HoldBall().

We turn now to the representation of state used by the keepers, ultimately for value function approximation as described in the next section. Note that values are only needed on the SMDP steps, and on these one of the keepers is always in possession of the ball. On these steps the keeper determines a set of state variables, computed based on the positions of: the keepers K_1–K_n, ordered as above; the takers T_1–T_m (m is the number of takers), ordered by increasing distance from K_1; and C, the center of the playing region (see Figure 2 for an example with 3 keepers and 2 takers). Let $dist(a, b)$ be the distance between a and b and $ang(a, b, c)$ be the angle between a and c with vertex at b. For example, $ang(K_3, K_1, T_1)$ is shown in Figure 2. With 3 keepers and 2 takers, we used the following 13 state variables:

- $dist(K_1, C)$, $dist(K_2, C)$, $dist(K_3, C)$
- $dist(T_1, C)$, $dist(T_2, C)$
- $dist(K_1, K_2)$, $dist(K_1, K_3)$
- $dist(K_1, T_1)$, $dist(K_1, T_2)$
- $\text{Min}(dist(K_2, T_1), dist(K_2, T_2))$
- $\text{Min}(dist(K_3, T_1), dist(K_3, T_2))$
- $\text{Min}(ang(K_2, K_1, T_1), ang(K_2, K_1, T_2))$
- $\text{Min}(ang(K_3, K_1, T_1), ang(K_3, K_1, T_2))$

This list generalizes naturally to additional keepers and takers, leading to a linear growth in the number of state variables.

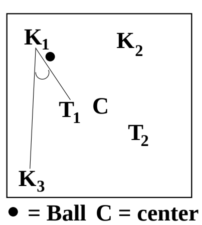

Figure 2: The state variables used for learning with 3 keepers and 2 takers. Keepers and takers are numbered by increasing distance from K_1, the keeper with the ball.

3.2 Takers

The takers are relatively simple, choosing only options of minimum duration (one step, or as few as possible given server misses) that exactly mirror CMUnited skills. When a taker has the ball, it tries to maintain possession by invoking HoldBall() for a step. Otherwise, it chooses an option that invokes one of {GoToBall(), BlockPass(K_2), BlockPass(K_3), ..., BlockPass(K_n)} for one step or as few steps as permitted by the server. In case no keeper has the ball (e.g., during a pass), K_1 is defined here as the keeper predicted to next gain possession by a routine from the CMUnited-99 team. We used the following three policies as taker benchmarks, characterized by their behavior when not in possession:

Random-T: Choose randomly from the n options, each with probability $\frac{1}{n}$.

All-to-ball: Always choose the GoToBall() option.

Hand-coded-T:

If fastest taker to the ball, or closest or second closest taker to the ball: choose the GoToBall() option;

Else let k be the keeper with the largest angle with vertex at the ball that is clear of takers: choose the BlockPass(k) option.

The takers' state variables are similar to those of the keepers. As before, C is the center of the region. T_1 is the taker that is computing the state variables, and T_2–T_m are the other takers ordered by increasing distance from K_1. $K_i$mid is the midpoint of the line segment connecting K_1 and K_i for $i \in [2, n]$ and where the K_i are ordered based on increasing distance of $K_i$mid from T_1. That is, $\forall i, j$ s.t. $2 \leq i < j$, $dist(T_1, K_i\text{mid}) \leq dist(T_1, K_j\text{mid})$. With 3 keepers and 3 takers, we used the following 18 state variables:

- $dist(K_1, C)$, $dist(K_2, C)$, $dist(K_3, C)$
- $dist(T_1, C)$, $dist(T_2, C)$, $dist(T_3, C)$
- $dist(K_1, K_2)$, $dist(K_1, K_3)$
- $dist(K_1, T_1)$, $dist(K_1, T_2)$, $dist(K_1, T_3)$
- $dist(T_1, K_2\text{mid})$, $dist(T_1, K_3\text{mid})$
- $\text{Min}(dist(K_2\text{mid}, T_2), dist(K_2\text{mid}, T_3))$
- $\text{Min}(dist(K_3\text{mid}, T_2), dist(K_3\text{mid}, T_3))$
- $\text{Min}(ang(K_2, K_1, T_2), ang(K_2, K_1, T_3))$
- $\text{Min}(ang(K_3, K_1, T_2), ang(K_3, K_1, T_3))$
- number of takers closer to the ball than T_1

Once again, this list generalizes naturally to different numbers of keepers and takers.

4. Reinforcement Learning Algorithm

We used the SMDP version of the Sarsa(λ) algorithm with linear tile-coding function approximation (also known as CMACs) and replacing eligibility traces (see Albus, 1981; Rummery & Niranjan, 1994; Sutton & Barto, 1998). Each player learned independently from its own actions and its own perception of the state. One complication is that most descriptions of Sarsa(λ) assume the agent has control and occasionally calls the environment to obtain the next state and reward, whereas here the RoboCup simulator retains control and occasionally presents state perceptions and option choices to the agent. This alternate orientation requires a different perspective on the standard algorithm. We need to specify three routines: 1) `RLstartEpisode`, to be run by the player on the first time step in each episode in which it chooses an option, 2) `RLstep`, to be run on each SMDP step, and 3) `RLendEpisode`, to be run when an episode ends. These three routines are given in Figure 3. Note that the traces decay only on SMDP time steps. In effect, we are using variable λ (Sutton & Barto, 1998), setting $\lambda = 1$ for these missing time steps. This scheme is one reasonable way to handle eligibility traces for SMDPs.

As we explain below, the primary memory vector $\vec{\theta}$ and the eligibility trace vector $\vec{e}$ are both of large dimension (e.g., thousands of dimensions for 3v2 keepaway), whereas the feature sets $\mathcal{F}_a$ are relatively small (e.g., 416 elements). The steps of greatest computational expense are those in which $\vec{\theta}$ and $\vec{e}$ are updated. By keeping track of the few nonzero components of $\vec{e}$, however, this expense can be kept to a small multiple of the size of the $\mathcal{F}_a$ (i.e., of 416). The initial value for $\vec{\theta}$ was $\vec{0}$.

For the results described in this paper we used the following values of the scalar parameters: $\alpha = 0.125$, $\epsilon = 0.01$, and $\lambda = 0$. In previous work (Stone et al., 2001), we experimented systematically with a range of values for the step-size parameter. We varied α over negative powers of 2 and observed the classical inverted-U pattern, with fastest learning at an intermediate value of about $\alpha = 2^{-3} = 0.125$, which we use here. We also experimented informally with ϵ and λ. The value $\epsilon = 0.01$ appears sufficiently exploratory without significantly affecting final performance. The effect of varying λ is not yet clear, so in this paper we treat the simplest case of $\lambda = 0$.

RLstartEpisode:

For all $a \in \mathcal{A}$(current state):
- $\mathcal{F}_a \leftarrow$ set of tiles for a and current state
- $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$

$LastOption \leftarrow \begin{cases} \arg\max_a Q_a & \text{w/prob. } 1-\epsilon \\ \text{random option} & \text{w/prob. } \epsilon \end{cases}$

$LastOptionTime \leftarrow CurrentTime$

$\vec{e} = \vec{0}$

For all $i \in \mathcal{F}_{LastOption}$:
- $e(i) \leftarrow 1$

RLstep:

$r \leftarrow CurrentTime - LastOptionTime$

$\delta \leftarrow r - Q_{LastOption}$

For all $a \in \mathcal{A}$(current state):
- $\mathcal{F}_a \leftarrow$ set of tiles for a and current state
- $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$

$LastOption \leftarrow \begin{cases} \arg\max_a Q_a & \text{w/prob. } 1-\epsilon \\ \text{random option} & \text{w/prob. } \epsilon \end{cases}$

$LastOptionTime \leftarrow CurrentTime$

$\delta \leftarrow \delta + Q_{LastOption}$

$\vec{\theta} \leftarrow \vec{\theta} + \alpha\,\delta\,\vec{e}$

$Q_{LastOption} \leftarrow \sum_{i \in \mathcal{F}_{LastOption}} \theta(i)$

$\vec{e} \leftarrow \lambda\vec{e}$

If player acting in current state:
- For all $a \in \mathcal{A}$(current state) s.t. $a \neq LastOption$:
  - For all $i \in \mathcal{F}_a$:
    - $e(i) \leftarrow 0$
- For all $i \in \mathcal{F}_{LastOption}$:
  - $e(i) \leftarrow 1$

RLendEpisode:

$r \leftarrow CurrentTime - LastOptionTime$

$\delta \leftarrow r - Q_{LastOption}$

$\vec{\theta} \leftarrow \vec{\theta} + \alpha\,\delta\,\vec{e}$

Figure 3: The three main routines of our Sarsa(λ) implementation presented for a keeper. A taker has the sign of the reward, r, reversed. As discussed in the text, the set of actions available, $\mathcal{A}$, may depend on the state. For example, the keepers not in possession of the ball must select the Receive option, whereas the keeper with the ball chooses from among HoldBall and Pass$k$ThenReceive.

It remains to specify how the feature sets $\mathcal{F}_a$ are constructed by the tile coding process. We use general tile coding software which allows us to take arbitrary groups of continuous state variables and lay infinite, axis-parallel tilings over them (e.g., see Figure 4). The tiles containing the current state in each tiling together make up a feature set $\mathcal{F}_a$, each action a indexing the tilings differently. The tilings are formally infinite in extent, but in our case, all the state variables are in fact bounded. Nevertheless, the number of possible tiles is extremely large, only a relatively few of which are ever visited (in our case about 10,000). Thus the primary memory vector $\vec{\theta}$ and the eligibility trace vector $\vec{e}$ have only this many nonzero elements. Using open-addressed hash-coding, only these nonzero elements need be stored.

In our experiments we used primarily single-dimensional tilings, i.e., simple stripes or intervals along each state variable individually. For each variable, 32 tilings were overlaid, each offset from the others by 1/32 of a tile width. In each tiling, the current state is in exactly one tile. The set of all these "active" tiles, one per tiling and 32 per group of state variables, is what makes up the $\mathcal{F}_a$. In the 3v2 case, there are 416 tiles in each $\mathcal{F}_a$ because there are thirteen state variables making thirteen single-variable groups, or $13 * 32 = 416$ total. For each state variable, we specified the width of its tiles based on the width of

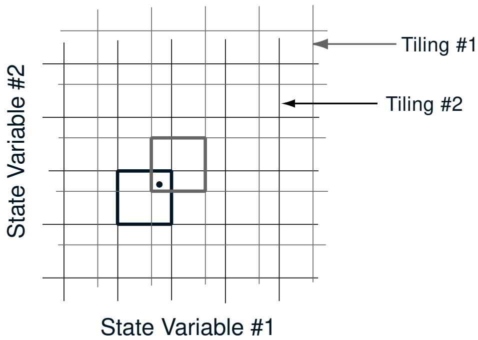

Figure 4: Our tile-coding feature sets were formed from multiple overlapping tilings of the state variables. Here we show two grid-tilings overlaid over the space formed by two state variables. (In this paper we primarily considered one-dimensional tilings.) Any state, such as that shown by the dot, is in exactly one tile of each tiling. Tile coding, also known as CMACs, has been widely used in conjunction with reinforcement learning systems (e.g., Watkins, 1989; Lin & Kim, 1991; Dean et al., 1992).

generalization that we desired. For example, distances were given widths of about 3.0 meters, whereas angles were given widths of about 10.0 degrees.

The choice here of state variables, widths, groupings, and so on, was done manually. Just as we as people have to select the state variables, we also have to determine how they are represented to the learning algorithm. A long-term goal of machine learning is to automate representational selections, but to date this is not possible even in supervised learning. Here we seek only to make the experimentation with a variety of representations relatively easy for us to do. The specific choices described here were made after some experimentation with learning by the keepers in a policy-evaluation scenario (Stone et al., 2001).

As described in Section 3.2, we used similar features and tilings for the takers as described above for the keepers. One exception is that we considered the last variable—the number of takers closer to the ball than T_1—as special in that it is effectively conjoined with each other state variable. We accomplished these conjunctions by creating separate tilings for each state variable and for each possible value of this special variable (it can take on as many values as there are takers). To counter the potential increase in computational complexity caused by extra tiling groups, we used only 8 tilings per conjunction, rather than 32.

This representation, which ignores most possible state-variable conjunctions, may be less well suited to the takers than it is to the keepers. We leave experimentation with representations for the takers as future work.

5. Empirical Results

Our main results to date have focused on learning by the keepers in 3v2 keepaway in a 20x20 region. For the opponents (takers) we used the Hand-coded-T policy (note that with just 2 takers, this policy is identical to All-to-ball). To benchmark the performance of the learned keepers, we first ran the three benchmark keeper policies, Random, Hold, and Hand-coded, as laid out in Section 3.1. Average episode lengths for these three policies were 5.5, 4.8, and 5.6 seconds respectively. Figure 5 shows histograms of the lengths of the episodes generated by these policies.

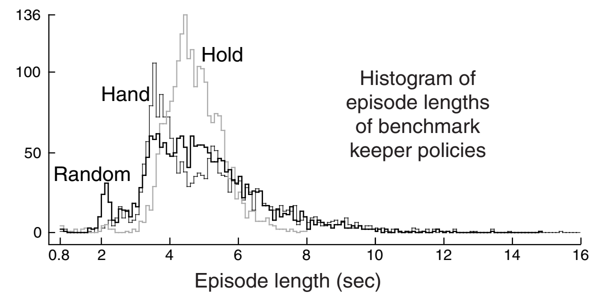

Figure 5: Histograms of episode lengths for the 3 benchmark keeper policies in 3v2 keepaway in a 20x20 region.

We then ran a series of eleven runs with learning by the keepers against the Hand-coded-T takers. Figure 6 shows learning curves for these runs. The y-axis is the average time that the keepers are able to keep the ball from the takers (average episode length); the x-axis is training time (simulated time $\approx$ real time). The performance levels of the benchmark keeper policies are shown as horizontal lines.

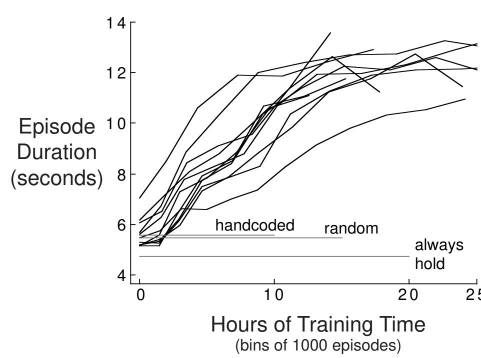

Figure 6: Multiple successful runs under identical characteristics: 3v2 keepaway in a 20x20 region against hand-coded takers.

This data shows that we were able to learn policies that were much better than any of the benchmarks. All learning runs quickly found a much better policy than any of the benchmark policies, including the hand-coded policy. A better policy was often found even in the first data point, representing the first 1000 episodes of learning. Qualitatively, the keepers appear to quickly learn roughly how long to hold the ball, and then gradually learn fine distinctions regarding when and to which teammate to pass.

Next, we applied the same algorithm to learn policies specialized for different region sizes. In particular, we trained the takers in 15x15 and 25x25 regions without changing any of the other parameters, including the representation used for learning. The results shown in Table 5 indicate that the learned policies again outperformed all of the benchmarks. In addition, it is apparent that policies trained in the same size region as they are tested (shown in boldface), performed better than policies trained in other region sizes. In general it is easier for the keepers to keep the ball in a larger field since the takers have further to run. Thus, we observe a general increase in possession time from left to right in the table. These results indicate that different policies are best on different field sizes. Thus, were we to take the time to laboriously fine-tune a hand-coded behavior, we would need to repeat the process on each field size. On the other hand, the same learning mechanism is able to find successful policies on all three field sizes without any additional human effort.

		Testing Field Size		
Keepers		15x15	20x20	25x25
Trained on field of size	15x15	**11.0**	9.8	7.2
	20x20	10.7	**15.0**	12.2
	25x25	6.3	10.4	**15.0**
Benchmarks	Hand	4.3	5.6	8.0
	Hold	3.9	4.8	5.2
	Random	4.2	5.5	6.4

Table 1: Performance (possession times) of keepers trained with various field sizes (and benchmark keepers) when tested on fields of various sizes.

Finally, we applied our learning algorithm to learn policies for a slightly larger task, 4 vs. 3 keepaway. Figure 7 shows that the keepers learned a policy that outperformed all of our benchmarks in 4v3 keepaway in a 30x30 region. In this case, the learning curves still appear to be rising after 40 hours: more time may be needed to realize the full potential of learning.

We have also begun exploring taker learning. As noted in Section 3.2, the representation used for the keepers may not be suited to taker learning. Nonetheless, using this representation, takers were able to learn policies that outperformed the Random and All-to-ball taker policies. The best learned policies discovered so far perform roughly equivalently to the Hand-coded-T policy.

Our on-going research is aimed at improving the ability of the takers to learn by altering their representation and/or learning parameters. One promising line of inquiry is into the efficacy of alternately training the takers and the keepers against each other so as to improve both types of policies.

Figure 7: Multiple successful runs under identical characteristics: 4v3 keepaway in a 30x30 region against hand-coded takers.

6. Related Work

Reinforcement learning has been previously applied to robotic soccer. Using real robots, Uchibe (1999) used reinforcement learning methods to learn to shoot a ball into a goal while avoiding an opponent. This task differs from keepaway in that there is a well-defined goal state. Also using goal-scoring as the goal state, TPOT-RL (Stone & Veloso, 1999) was successfully used to allow a full team of agents to learn collaborative passing and shooting policies using a Monte Carlo learning approach, as opposed to the TD learning explored in this paper. Andou's (1998) "observational reinforcement learning" was used for learning to update players' positions on the field based on where the ball has previously been located.

Perhaps most related to the work reported here, Riedmiller et al. (2001) uses reinforcement learning to learn low-level skills ("moves"), such as kicking, ball-interception, and dribbling, as well as a cooperative behavior in which 2 attackers try to score against one or two takers. In contrast to our approach, this work use the full sensor space as the input representation, with a neural network used as a function approximator. The taker behaviors were always fixed and constant, and no more than 2 attackers learned to cooperate.

Distributed reinforcement learning has been explored previously in discrete environments, such as the pursuit domain (Tan, 1993) and elevator control (Crites & Barto, 1996). This latter task differs in that the domain is continuous, that it is real-time, and that there is noise both in agent actions and in state-transitions.

In conjunction with the research reported here, we have explored techniques for keepaway in a full 11 vs. 11 scenario played on a full-size field (McAllester & Stone, 2001). The successful hand-coded policies were incorporated into ATT-CMUnited-2000, the 3rd-place finisher in the RoboCup-2000 simulator competition.

Acknowledgements

We thank Satinder Singh for his assistance in formulating our reinforcement learning approach to keepaway; Patrick Riley and Manuela Veloso for their participation in the creation of the CMUnited-99 agent behaviors that were used as a basis for our agents; and Tom Miller and others at the University of New Hampshire for making available their CMAC code.

References

Albus, J. S. (1981). *Brains, behavior, and robotics.* Peterborough, NH: Byte Books.

Andou, T. (1998). Refinement of soccer agents' positions using reinforcement learning. In H. Kitano (Ed.), *RoboCup-97: Robot soccer world cup I*, 373–388. Berlin: Springer Verlag.

Baird, L. C., & Moore, A. W. (1999). Gradient descent for general reinforcement learning. *Advances in Neural Information Processing Systems* (pp. 968–974). The MIT Press.

Bradtke, S. J., & Duff, M. O. (1995). Reinforcement learning methods for continuous-time Markov decision problems. *Advances in Neural Information Processing Systems 7* (pp. 393–400). San Mateo, CA: Morgan Kaufmann.

Crites, R. H., & Barto, A. G. (1996). Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems 8* (pp. 1017–1023). Cambridge, MA: MIT Press.

Dean, T., Basye, K., & Shewchuk, J. (1992). Reinforcement learning for planning and control. In S. Minton (Ed.), *Machine learning methods for planning and scheduling*, 67–92. Morgan Kaufmann.

Gordon, G. (2001). Reinforcement learning with function approximation converges to a region. *Advances in Neural Information Processing Systems* (pp. 1040–1046). The MIT Press.

Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osawa, E., Matsubara, H., Noda, I., & Asada, M. (1997). The RoboCup synthetic agent challenge 97. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (pp. 24–29). San Francisco, CA: Morgan Kaufmann.

Lin, C.-S., & Kim, H. (1991). CMAC-based adaptive critic self-learning control. *IEEE Trans. Neural Networks* (pp. 530–533). IEEE Press.

McAllester, D., & Stone, P. (2001). Keeping the ball from CMUnited-99. In P. Stone, T. Balch and G. Kraetszchmar (Eds.), *RoboCup-2000: Robot soccer world cup IV*, 333–338. Berlin: Springer Verlag.

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

Puterman, M. L. (1994). *Markov decision processes.* NY: Wiley.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* San Mateo, CA: Morgan Kaufmann.

Riedmiller, M., Merke, A., Meier, D., Hoffman, A., Sinner, A., Thate, O., & Ehrmann, R. (2001). Karlsruhe brainstormers—a reinforcement learning approach to robotic soccer. In P. Stone, T. Balch and G. Kraetszchmar (Eds.), *RoboCup-2000: Robot soccer world cup IV*, 367–372. Berlin: Springer Verlag.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Technical Report CUED/F-INFENG/TR 166). Cambridge University Engineering Department.

Stone, P. (2000). *Layered learning in multiagent systems: A winning approach to robotic soccer.* MIT Press.

Stone, P., & McAllester, D. (2001). An architecture for action selection in robotic soccer. *Proceedings of the Fifth International Conference on Autonomous Agents* (pp. 316–323). New York, NY: ACM Press.

Stone, P., Sutton, R. S., & Singh, S. (2001). Reinforcement learning for 3 vs. 2 keepaway. In P. Stone, T. Balch and G. Kraetszchmar (Eds.), *RoboCup-2000: Robot soccer world cup IV*, 249–258. Berlin: Springer Verlag.

Stone, P., & Veloso, M. (1999). Team-partitioned, opaque-transition reinforcement learning. In M. Asada and H. Kitano (Eds.), *RoboCup-98: Robot soccer world cup II*. Berlin: Springer Verlag. Also in *Proceedings of the Third International Conference on Autonomous Agents*, 1999.

Sutton, R., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems 8* (pp. 1038–1044). Cambridge, MA: MIT Press.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. *Proceedings of the Tenth International Conference on Machine Learning* (pp. 330–337). Morgan Kaufmann.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*, 674–690.

Uchibe, E. (1999). *Cooperative behavior acquisition by learning and evolution in a multi-agent environment for mobile robots*. Doctoral dissertation, Osaka University.

Veloso, M., Stone, P., & Bowling, M. (1999). Anticipation as a key for collaboration in a team of agents: A case study in robotic soccer. *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II* (pp. 134–143). Bellingham, WA: SPIE.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Doctoral dissertation, King's College, Cambridge, UK.