Keepaway Soccer: From Machine Learning Testbed to Benchmark

Peter Stone, Gregory Kuhlmann, Matthew E. Taylor, and Yaxin Liu

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188
{pstone,kuhlmann,mtaylor,yxliu}@cs.utexas.edu

Abstract. Keepaway soccer has been previously put forth as a *testbed* for machine learning. Although multiple researchers have used it successfully for machine learning experiments, doing so has required a good deal of domain expertise. This paper introduces a set of programs, tools, and resources designed to make the domain easily usable for experimentation without any prior knowledge of RoboCup or the Soccer Server. In addition, we report on new experiments in the Keepaway domain, along with performance results designed to be directly comparable with future experimental results. Combined, the new infrastructure and our concrete demonstration of its use in comparative experiments elevate the domain to a machine learning *benchmark*, suitable for use by researchers across the field.

1 Introduction

Keepaway soccer in the Soccer Server used at RoboCup has been previously put forth as a testbed for machine learning [15]. Since then it has been used for research on temporal difference reinforcement learning with function approximation [16], evolutionary learning [12], relational reinforcement learning [20], and behavior transfer [18].

These successes notwithstanding, the domain has remained inaccessible to many potential users due to the considerable implementation effort required. In particular, though the *domain* itself is publically available, it has been necessary for each researcher to build up *agents* capable of following the rules of Keepaway and executing the required low-level behaviors such as intercepting a moving ball, passing the ball, and moving to open space. Due to the complexity of the simulator, building up such agents is no simple task. As a result, researchers who have no prior experience in RoboCup, or who are unwilling to invest considerable start-up effort, have not been able to use the testbed.

Even if willing to invest this effort, the resulting experiments are not directly comparable with previously published results. Because each experiment is done with different underlying agents, the performance results may vary more from inter-agent differences than from the learning algorithms under study.

These two barriers — inaccessibility to the non-domain-experts and incomparability of results across studies — have prevented the Keepaway *testbed* from serving as a *benchmark* for the machine learning community. This paper reports on new resources that elevate the Keepaway testbed to a benchmark problem for machine learning. In particular, it describes a repository that:

1. Provides standard, open source implementations for all aspects of the problem except the learning algorithm itself;
2. Provides a step-by-step tutorial for non-domain-experts to get up to speed easily; and
3. Provides the graphical tools necessary to evaluate progress.

In addition, we report concrete numerical results for several easily replicable approaches using the material from the repository. These numerical results are designed to be directly comparable with future experimental studies.

Finally, we illustrate the use of this benchmark by presenting an empirical study of different function approximators used within a temporal difference learning approach to the problem. Previous results indicated that using a form of linear tile-coding (a CMAC [1]) to approximate the value function led to results that were better than hand-coded approaches. However, it was not known to what extent the CMAC itself was responsible for these results. In this paper we directly compare the CMAC against other function approximators, including a variation of CMAC based on radial basis functions, as well as neural networks.

2 Background

This section introduces the Keepaway task, surveys previous examples of learning in this domain, and specifies a standardized learning scenario to be used as a machine learning benchmark.

2.1 The Keepaway Task

Keepaway is a subproblem of RoboCup simulated soccer in which one team, the *keepers*, tries to maintain possession of the ball within a limited region, while the opposing team, the *takers*, attempts to gain possession [15]. 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 an episode with 3 keepers and 2 takers (called 3 vs. 2, or 3v2 for short) playing in a $20m \times 20m$ region.[1]

2.2 The Soccer Server

Since late 2002, the Keepaway task has been part of the official release of the open source Soccer Server used at RoboCup.[2] Agents in the simulator [11] receive visual perceptions every 150 *msec* indicating the relative distance and angle to visible objects in the world, such as the ball and other agents. They may execute a primitive, parameterized action such as `turn(`*angle*`)`, `dash(`*power*`)`, or

[1] Flash files illustrating the task are available at `http://www.cs.utexas.edu/~AustinVilla/sim/keepaway/`

[2] Starting with version 9.1.0

`kick`(*power*,*angle*) every 100 *msec*. Thus the agents must sense and act asynchronously. Random noise is injected into all sensations and actions. Individual agents must be controlled by separate processes, with no inter-agent communication permitted other than via the simulator itself, which enforces communication bandwidth and range constraints. Full details of the simulator are presented in the server manual [6].

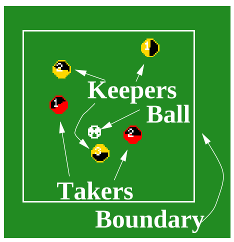

Fig. 1: A screen shot from the middle of a 3 vs. 2 Keepaway episode in a 20m x 20m region.

When started in a special mode, the simulator enforces the rules of the Keepaway task, as described above, instead of the rules of full soccer. In particular, the simulator places the players at their initial positions at the start of each episode and ends an episode when the ball leaves the play region or is taken away. In this mode, the simulator also informs the players when an episode has ended and produces a log file with the duration of each episode.

2.3 Previous Studies

Although the Keepaway task has been available in the server for some time, it required knowledge of player development to be useful as a machine learning testbed. Nonetheless, there are a few examples of machine learning research involving RoboCup Keepaway.

In addition to our own previous work [16, 18], work by DiPietro et al. [12] applied evolutionary algorithms to train 3 keepers against 2 takers in the Soccer Server. Other work by Walker et al. [20] used relational reinforcement learning to learn the value function for a keeper coordinating with 2 "smart" teammates against 2 takers.

There has been additional previous work in Keepaway using simulators other than the Soccer Server. Whiteson et al. [21] used neuroevolution to train keepers in the SoccerBots domain [3], an extension of the more abstract TeamBots simulator [2]. Also, Hsu and Gustafson [9] evolved keepers for 3 vs. 1 Keepaway in the TeamBots simulator.[3]

Clearly it is not possible to to directly compare performance of work using simulators with different primitive actions and different game dynamics. But even work within the same simulator cannot typically be compared directly because the approaches differ in their set of high-level behaviors, implementation of basic skills, fixed opponent policies, and sometimes even performance metrics. It is exactly this problem that our benchmark repository seeks to address.

2.4 Standardized Task

When Keepaway was introduced as a testbed [15], a standard task was defined. The main contribution of the current work is the infrastructure required to easily implement that task (Section 3). This section reviews the standardized task as previously formulated.

[3] Gustafson has made some code available that contributes to this more abstract simulator having some of our defined properties of a benchmark: `http://www.cs.nott.ac.uk/~smg/kas/keepaway-v0.01.html`

The Keepaway problem maps fairly directly onto the discrete-time, episodic, reinforcement-learning framework. As a way of incorporating domain knowledge, the learners choose not from the simulator's primitive actions but from a set of higher-level macro-actions implemented as part of the player. These macro-actions can last more than one time step, and the keepers have opportunities to make decisions only when an on-going macro-action terminates. To handle such situations, it is convenient to treat the problem as a semi-Markov decision process, or SMDP [13, 5]. The macro-actions used (and fully provided as a part of our repository) can be found in [16]. The agent can make decisions at discrete SMDP time steps, at which macro-actions are initiated and terminated.

For the purpose of defining a standardized task, we focus on training the keepers, but training the takers can be done similarly in the Keepaway domain, as suggested in [16]. The keepers learn in a constrained policy space. They have the freedom to decide which action to take only when in possession of the ball. A keeper in possession may either hold the ball or pass to one of its teammates. Keepers not in possession of the ball are required to execute the **Receive** macro-action in which the player who can get there the soonest goes to the ball and the remaining players try to get open for a pass.

For training the keepers, the behavior of the takers is "hard-wired" and relatively simple. The two takers that can get there the soonest go to the ball, while the remaining takers, if present, try to block open passing lanes. Similarly, for training the takers, the keepers may be hard-wired. These hard-wired behaviors are provided in the repository.

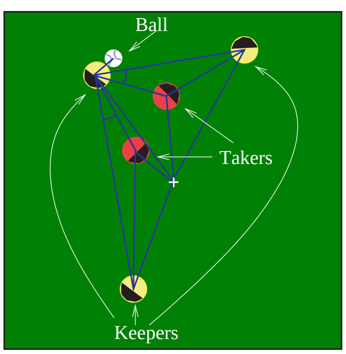

Fig. 2: This diagram depicts the 13 state variables used for learning with 3 keepers and 2 takers. There are 11 distances to players, the center of the field, and the ball, as well as 2 angles along passing lanes.

Also as a way of incoporating domain knowledge, the learners do not make decisions based on raw positional information but based on higher-level features that form the states for the Keepaway learning task. The keepers' states comprise distances and angles of the keepers $K_1, \ldots, K_n$, the takers $T_1, \ldots, T_m$, and the center of the playing region. Keepers and takers are ordered by increasing distance from the ball and when learning takes place, K_1 is always the keeper in possession of the ball. The 13 state variables for 3v2 Keepaway are (see Figure 2):

- Distances from players to center of region,
- Distances from other players to K_1,
- Distances from teammates to their closest opponent, and
- For each K_i ($i = 2, \ldots, n$), the minimal angle with the vertex at K_1 between K_i and an opponent.

We can easily vary the size of the Keepaway region, the number of keepers, and the number of takers to change the Keepaway task. Our framework provides a standard interface to the learner in terms of macro-actions, states, and rewards.

We choose episode duration as the performance measure for this task. The keepers attempt to maximize it while the the takers try to minimize it. To this end, it is natural to give the learners a constant positive reward for each time step an episode persists. Complete details on the task and the learning scenario can be found elsewhere [16].

3 Benchmark Repository

We have implemented a standardized Keepaway player framework in C++ and released its code base for public use in an online repository.[4] The code base provides an open source implementation for all aspects of the Keepaway problem except the learning algorithm itself, which is intended to be the object of study by each individual researcher.

3.1 Standardized Keepaway Player

We have created an implementation of a standardized player built upon the player framework developed by the UvA Trilearn team [8]. This framework handles communication and synchronization with the server, world model update, localization, and low- and mid-level skills.

On top of this framework, we added additional skills necessary for Keepaway and implemented the fixed policy carried out by the keepers when they do not have the ball. Also, we have implented hand-coded takers that follow the policy described previously.

By default, a player in the Soccer Server is only able to see objects in a 90-degree cone in front of them. A difficult problem in developing agents that are meant to coordinate in dynamic environments with limited vision is trying to maintain a correct distributed world model. This requires agents to decide where to look, what to communicate, whom to listen to, and how to incorporate second-hand information. Although we have previously demonstrated that it is possible to get learning to work in this scenario [10], the players do not perform at as high of a level. For this reason, the players operate with 360-degree (but still noisy) vision by default. A rudimentary implementation of communication and information-gathering actions is included, but is not recommended. We hope that in future development of the players, the level of play in the 90-degree case can be increased such that the 360-degree assumption is no longer necessary.

Learning Agent Interface The Keepaway player implementation is constructed in such a way that the details of the Keepaway domain are completely abstracted from the high-level action selection. This was done to allow new learning algorithms to be integrated into the players with minimal effort.

From the learning algorithm's perspective, the Keepaway problem is presented as a generic SMDP. The state is represented as a fixed-length vector of continuous values. A macro-action is represented as an integer ranging from 0 to $numActions - 1$. The reward is a single continuous value received after each macro-action terminates.

[4] `http://www.cs.utexas.edu/~AustinVilla/sim/keepaway/`

A new learning agent must implement the **SMDPagent** interface, which consists of the following three functions:

- **int startEpisode(double state[])**
  This function is called the first time this player has an action opportunity in an episode. In other words, it is called when the player first has possession of the ball. If the player never obtains the ball, this function will never be called. The agent is supplied with the current state and is expected to return a macro-action to be executed.
- **int step(double reward, double state[])**
  This function is called at every action opportunity for this player after the first one. The reward accumulated during the execution of the previous macro-action is given along with the new state. A macro-action is again expected to be returned.
- **void endEpisode(double reward)**
  This function is called when the player receives notice from the server that the episode has ended. The agent is supplied with the reward accumulated from the last macro-action up until the end of the episode. Note that this function is called after every episode, even when this player never touches the ball.

Although the players do not come with any learning code, a few fixed policies are supplied:

- **Random** - choose actions uniformly at random
- **Always Hold** - always choose the hold action
- **Hand-coded** - a simple handcoded policy that holds the ball when no takers are nearby and passes to the most open teammate otherwise. More details can be found in [16].

The known performance of these policies as reported later in this paper (see Section 4) serve as a sanity check for new installations of the system.

3.2 Tools

The player source code package comes with a set of scripts that are helpful in running experiments and analyzing the resulting data. There are scripts to start and stop the simulation. Another script can be executed during the simulation to launch the Soccer Monitor, the standard visualization tool for the simulator, in a special mode that displays the Keepaway region on field. Additional scripts use the known structure of the `.kwy` log files to compress and decompress them much more compactly than using a standard compression utility.

One of the most useful tools is a script that converts the episode duration data in a `.kwy` file into a representation that is appropriate for generating a learning curve using a tool such as *gnuplot*. This script uses a "sliding window" to find average episode durations for each fixed-sized sequence of episodes. Along with the size of the window (number of episodes), the script takes in an additional parameter that specifies the coefficient of a low-pass filter used for smoothing the curve. A gnuplot style file is also supplied to produce graphs similar to the ones included in this paper (e.g. Figure 4).

Finally, there is a tool to generate histograms of episode durations. The intended purpose of this tool is to allow someone to visualize the distribution of episode durations when evaluating a fixed policy. Again, a gnuplot style file is included for generating figures appropriate for publication.

3.3 Online Tutorials

In addition to source code itself, the web site repository contains a step-by-step tutorial of how to use the code. The goal of the tutorial is to allow for someone who is not an expert in the RoboCup simulated soccer domain to get up to speed easily.

The tutorial is divided into two sections. The first section walks through the necessary steps for downloading and installing the simulator and players, starting a simulation using one of the supplied hand-coded policies, and generating a learning curve. Two graduate students from our lab that had never worked in the Keepaway domain before were able to successfully complete this section in a matter of minutes.

The second part of the tutorial discusses how to incorporate a new learning algorithm into the provided player source code. We include skeleton code for download from within this section of the web site that can serve as the starting point for a new learning agent implementation. Thus, the main effort required on the part of a new user is exactly the porting of one's own learning approach to the place-holders within the provided source code.

4 An Empirical Study

Though many learning approaches are possible in this domain, we now consider a particular learning approach to learning Keepaway for the keepers. The keepers learn their task using episodic SMDP Sarsa(λ) [17, 16], a well-understood temporal difference algorithm and naturally fit into the **SMDPagent** interface.

The state variables are continuous and therefore suggest value function approximation. We consider episodic SMDP Sarsa(λ) but with different function approximators. A function approximator in a Sarsa(λ) learner maps states to a vector of state-action values, one entry for each action, and the Sarsa(λ) learner uses the state-action values to perform on-policy learning.

Static Keepaway Policies

Policy	3 vs. 2	4 vs. 3	5 vs. 4
Always Hold	**3.4**±1.5	**4.1**±1.8	**4.8**±2.2
Random	**7.5**±3.7	**8.3**±4.4	**9.5**±5.1
Hand-coded	**8.3**±4.7	**9.2**±5.2	**10.8**±6.7

Table 1: This table details the average possession time and standard deviation in seconds for three simple policies included in our distributed code on three different Keepaway tasks. 3 vs. 2 is run on a 20m x 20m field, 3 vs. 4 is run on a 25m x 25m field, and 5 vs. 4 is run on a 30m x 30m field. These numbers may be used as benchmarks to be compared against other learned policies.

Within this framework, a question that arises is the efficacy of different function approximators. By using an implementation of the SMDPagent interface we are able to easily substitute different function approximators, a key component of a value-based RL learning. In this section we compare three such function approximators, CMACs, a novel extension to CMACs using radial basis functions which we denote RBF networks, and neural networks, in the Sarsa approach to the Keepaway task. The training performs a version of Sarsa using function approximators [14], with adaptations for SMDPs.

In order to quantify the learning rates of different learning algorithms and function approximators, we analyze the `.kwy` files produced by the Soccer Server in Keepaway mode. To produce a learning curve we "window" the data so that every point on a learning graph is the average of 1000 Keepaway episodes. The noise in the sensors and actuators is large enough that the variance between different episodes is large, even within a single trial using a static policy. By averaging episodes over time we are able to reduce much of the noise in our graph and still show representative curves.

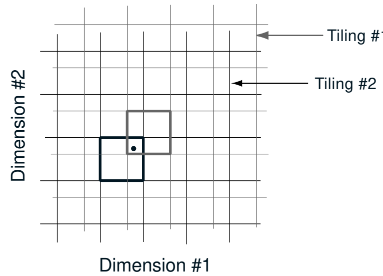

Fig. 3: The tile-coding feature sets are formed from multiple overlapping tilings. The state variables are used to determine the activated tile in each of the different tilings. Every activated tile then contributes a weighted value to the total output of the CMAC for the given state. Increasing the number of tilings allows the tile-coding to generalize better while decreasing the tile size allows more accurate representations of smaller details. Note that we primarily use one-dimensional tilings but that the principles apply in the n-dimensional case.

The possession times for the three static policies provided can be found in Table 1. Note that the standard deviation is a measure of the difference of windowed averages across trials, not a measure of the variation in episode lengths within a single trial. We will see that both function approximators examined allow players to learn an average possession time better than these three static policies. While past research [16] has shown that CMAC function approximation allows learning in the Keepaway domain, this is the first research showing comparable, or perhaps better, learning results, in this case using the new RBF function approximator.

4.1 CMAC Function Approximation

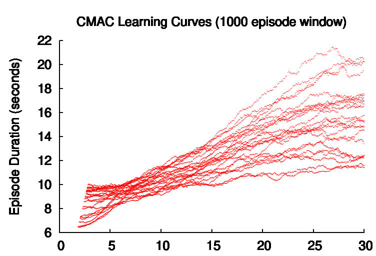

Fig. 4: This figure presents the learning curves for 24 independent 3 vs. 2 trials using a CMAC function approximator.

CMACs are a form of linear tile-coding function approximation that have been successfully used in many reinforcement learning systems [1]. CMACs allow us to take arbitrary groups of continuous state variables and lay infinite, axis-parallel tilings over them (see Figure 3). Using this method we are able to discretize the continuous state space by using tilings while maintaining the capability to generalize via multiple overlapping tilings. The number of tiles and width of the tilings are hardcoded and this dictates which state values will activate which tiles. The function approximation is learned by changing how much each tile contributes to the output of the function approximator. By default, all the CMAC's weights are initialized to zero. This approach to function approximation in the RoboCup soccer domain is detailed by Stone et al. [16].

CMACs have been used previously in the Keepaway domain [16]. We include results here as a point of comparison. In Figure 4 we see that the keepers learn to increase the time they are able to control the ball through training. The average learned possession time over 24 trials after 30 simulator hours of training is 15.69 seconds with a standard deviation of 2.81 seconds. Again, the average possession time for each trial is averaged over 1000 episodes to reduce the impact of noise.

4.2 RBF Function Approximation

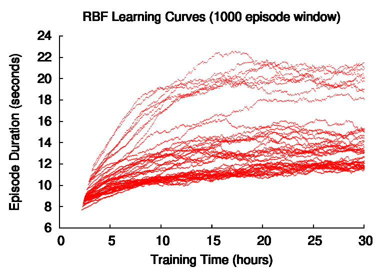

Fig. 5: This figure presents the learning curves for 40 independent 3 vs. 2 trials using an RBF function approximator.

A radial basis function (RBF) is a generalization of the tile coding idea to a continuous function [17]. In the one-dimensional case, an RBF approximator is a linear function approximator $\hat{f}(x) = \sum_i w_i f_i(x)$, where the basis functions have the form $f_i(x) = \phi(|x - c_i|)$, x is the current state, and c_i is the center for feature i. A CMAC is a special type of RBF approximator with c_i's equally spaced and $\phi(x)$ a step function. Here we use Gaussian radial basis functions, where $\phi(x) = \exp(-\frac{x^2}{2\sigma^2})$, and the same c_i's as a CMAC. The learning for RBF networks are identical to that for CMACs except for the calculation of state-action values where the RBFs are used. As is the case for CMACs, the state-action values are computed as a sum of one-dimensional RBFs, one for each feature. By tuning σ, the experimenter can control the width of the Gaussian function and therefore the amount of generalization over the state space. In our implementation, a value of $\sigma = 1.0$ creates a Gaussian which roughly spans 9 CMAC tiles, a value of $\sigma = 0.5$ spans 5 tiles, and $\sigma = 0.25$ activates roughly 3 tiles. We found the value of $\sigma = 0.25$ to perform the best, but more tuning would possibly produced better results than reported here.

In Figure 5 we see keepers can successfully learn to keep the ball with an RBF function approximator for successively longer periods of time after training. The average learned possession time over 40 trials after 30 simulator hours of training is 14.23 seconds with a standard deviation of 3.14 seconds.

By comparing Figures 4 and 5, we note that the RBF trials appear to be learning faster: they have better performance between the 5-hour and 25-hour marks (though not significantly different average performance). The best-case performance of RBF at the 30-hour mark is comparable to that of CMAC, but more RBF trials land at the worst-case end.

4.3 Neural Network Function Approximation

Feedforward neural networks are a very popular type of function approximator and have had some notable past successes in reinforcement learning [7, 19]. We use three seperate 13-20-1 networks, one for each action. Inputs to the neural network are set to the value of state variables and each network's output corresponds to an action. Nodes in the hidden layer have a sigmoid transfer function and output nodes are linear. We use standard backpropagation to modify the weights in the neural networks to back up Sarsa(λ) values.

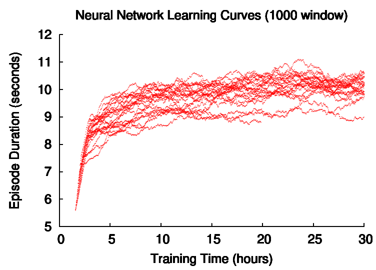

Fig. 6: This figure presents the learning curves for 20 independent 3 vs. 2 trials using a neural network function approximator.

In Figure 6 we see keepers also learn to keep the ball with a neural network function approximator for longer periods of time after training. The average learned possession time over 20 trials after 30 simulator hours of training is 10.13 seconds with a standard deviation of 0.29 seconds.

By comparing Figure 6 to Figures 4 and 5, we note that neural networks do not learn as fast as CMACs or RBFs, but have a lower variance. However, we did not try to optimize the neural networks' parameters and ran fewer experiments relative to the CMACs and RBFs. Neural networks can theoretically have better performance: they can encode arbitrarily complex interactions among state variables while CMACs and RBFs in our implementation only combine effects of different state variables in a simple way (summation).

5 Future Work

In the Section 4 we give an example study comparing the efficacy of two different function approximators using the Keepaway benchmark. Other function approximators and learning algorithms can be easily inserted into the testbed and directly compared to these two function approximators, allowing experimenters to quickly test the relative benefits of different techniques.

The infrastructure is of course not limited to function approximation studies. Different reinforcement learning algorithms, both value-based and policy search, may be compared. Evolutionary methods (as in [12]) may be evaluated and compared directly to temporal difference methods; different temporal difference methods may be directly compared (such as Sarsa vs. Q-learning as in [16]). Multi-agent learning questions may be addressed, such as how learning rates are affected when opponents or teammates learn simultaneously with an agent, learn sequentially, or do not learn at all. Different state representations can be easily expressed, enabling the investigation of state abstraction. Alternate actions can be implemented, allowing the investigation of hierarchical RL questions such as using options to speed up learning. We anticipate exploring some of these areas in the future using this infrastructure.

Our Keepaway benchmark infrastructure provides an easy way of consistently comparing the relative performance of different methods when investigating any of the previously mentioned research questions. By using the same benchmark platform, researchers will be able to make quantitative comparisons between different learning methods. To that end we will make every effort to not make changes to the infrastructure that affect performance, thus making direct comparisons across versions of our implementation legitimate. However, we intend to make improvements to the communication and distributed sensing code for use with limited vision. Because of this and other possible changes, when reporting results, we encourage researchers to always cite the implementation version number as well as any non-default settings that were used. Doing so will ensure the validity of direct comparisons as well as enable the repeatability of all experiments.

6 Conclusion

This paper makes two main contributions. First and foremost, it introduces a source code repository including a set of tools and tutorials designed to enable all machine learning researchers to use the Keepaway soccer domain as a benchmark task. Second, it introduces a function approximation method not previously tested in this domain and empirically evaluates it on this benchmark task. The RBF network performs at least as well as, and perhaps a little better than, the previously-used CMAC method.

While there is an established repository for supervised machine learning benchmarks [4], there is currently no comparable repository of sequential decision-making tasks. For this reason, releasing our new benchmark provides a valuable service. We plan to continue maintaining the repository at `http://www.cs.utexas.edu/~AustinVilla/sim/keepaway/` for the foreseeable future. It is our hope that it will be a benefit to both the RoboCup and the machine learning communities. For people already involved in RoboCup, it standarizes a benchmark for machine learning research within the domain, and could serve as a domain of interest for the Special Interest Group (SIG) on multiagent learning. [5] For the machine learning community it makes experimentation within the RoboCup domain accessible to everyone.

Acknowledgments

This research was supported in part by NSF CAREER award IIS-0237699, ONR YIP award N00014-04-1-0545, and DARPA grant HR0011-04-1-0035.

References

1. J. S. Albus. *Brains, Behavior, and Robotics*. Byte Books, Peterborough, NH, 1981.
2. Tucker Balch. Teambots, 2000. `http://www.teambots.org`.
3. Tucker Balch. Teambots domain: Soccerbots, 2000. `http://www-2.cs.cmu.edu/~trb/TeamBots/Domains/SoccerBots`.
4. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
5. S. J. Bradtke and M. O. Duff. Reinforcement learning methods for continuous-time Markov decision problems. In T. Leen G. Tesauro, D. Touretzky, editor, *Advances in Neural Information Processing Systems 7*, pages 393–400, San Mateo, CA, 1995. Morgan Kaufmann.
6. Mao Chen, Ehsan Foroughi, Fredrik Heintz, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Patrick Riley, Timo Steffens, Yi Wang, and Xiang Yin. Users manual: RoboCup soccer server manual for soccer server version 7.07 and later, 2003. Available at `http://sourceforge.net/projects/sserver/`.

[5] `http://sserver.sourceforge.net/SIG-learn/`

7. Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, Cambridge, MA, 1996. MIT Press.
8. Remco de Boer and Jelle R. Kok. The incremental development of a synthetic multi-agent system: The uva trilearn 2001 robotic soccer simulation team. Master's thesis, University of Amsterdam, The Netherlands, February 2002.
9. W. H. Hsu and S. M. Gustafson. Genetic programming and multi-agent layered learning by reinforcements. In *Genetic and Evolutionary Computation Conference*, New York,NY, July 2002.
10. Gregory Kuhlmann and Peter Stone. Progress in learning 3 vs. 2 keepaway. In *Proceedings of the RoboCup-2003 Symposium*, Padova, Italy, July 2003.
11. Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.
12. Anthony Di Pietro, Lyndon While, and Luigi Barone. Learning in RoboCup keepaway using evolutionary algorithms. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1065–1072, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
13. M. L. Puterman. *Markov Decision Processes*. Wiley, NY, 1994.
14. G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.
15. Peter Stone and Richard S. Sutton. Keepaway soccer: a machine learning testbed. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup-2001: Robot Soccer World Cup V*, pages 214–223. Springer Verlag, Berlin, 2002.
16. Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 2005. To appear.
17. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
18. Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, July 2005. To appear.
19. Gerald Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.
20. T. Walker, J. Shavlik, and R. Maclin. Relational reinforcement learning via sampling the space of first-order conjunctive features. In *Proceedings of the ICML Workshop on Relational Reinforcement Learning*, Banff, Canada, July 2004.
21. Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone. Evolving keepaway soccer players through task decomposition. *Machine Learning*, 59(1):5–30, May 2005.