A Task Specification Language for Bootstrap Learning (Extended Abstract)

Ian Fasel, Michael Quinlan, Peter Stone
Department of Computer Sciences, The University of Texas at Austin
{ianfasel,mquinlan,pstone}@cs.utexas.edu

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents*

General Terms

Human Factors, Standardization, Languages

Keywords

Reinforcement Learning, Human Computer Interaction

1. INTRODUCTION

Traditionally, research in the reinforcement learning (RL) community has been devoted to developing domain-independent algorithms such as SARSA [13], Q-learning [16], prioritized sweeping [8], or LSPI [6], that are designed to work for any given state space and action space. However, the *modus operandi* in RL research has been for a human expert to re-code each learning environment, including defining the actions and state features, as well as specifying the algorithm to be used. Typically each new RL experiment is run by explicitly calling a new program (even when learning can be biased by previous learning experiences, as in transfer learning [10, 15, 14]). Thus, while standards have developed for describing and testing individual RL algorithms (e.g., RL-Glue [17]), no such standards have developed for the problem of describing complete tasks to a preexisting agent.

In this paper we present a new language for specifying complete tasks, and a framework for agents to learn a new policy for solving these tasks. This language was designed for the large, multi-year, multi-institution "Bootstrap Learning" (BL) project [1], which aims to enable humans to teach agents multiple different tasks using different instructional techniques or sources of training data. Our "BL Task Learning" (or BLTL) language, specific for sequential decision making tasks, allows the human teacher to specify to the agent starting states, reward functions, termination conditions, *advice* [7, 5] or *demonstrations* [9], indicate relevant previous experience (enabling *transfer learning* [10, 15, 14]), use previously taught tasks as primitive actions in new tasks, or specify portions of a more complex task which are to be refined by learning (enabling *task decomposition*[2, 3, 12]). Because *parts* of larger tasks can be learned each with a different technique, we enable multiple learning methods to be synergistically integrated in a single agent that is far more capable than an agent using any one learning algorithm.

Cite as: A Task Specification Language for Bootstrap Learning, (Extended Abstract), Ian Fasel, Michael Quinlan, Peter Stone, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 1169–1170

2. LANGUAGE PRIMITIVES

The following list of functions with informal descriptions represents our language for teacher-student interaction regarding sequential decision making problems.

BeginTaskDescription("name") Prepare to start learning a task, `name` defines the new policy.
Environment("simulator" or "function") Can be a `simulator`, or simply use transition `function` directly.
BeginEpisode("world state") Episodes to begin by initializing simulator to `world state`.
OnEpisodeEnd("function") On episode end issue simulator specific `function` e.g. `RestartFromConfigurationX`
StepReward("function") Let `function` be the reward function at each timestep.
AddToStateSpace("function") Concatenate `function` result to the list of state space variables.
AddToActionSpace("function") Add to the list of possible actions. Could be primitive or complex options.
WrapperPolicy("policy", "condition") For task decomposition: run pre-existing policy `policy`, but switch to currently learning policy when `condition` is `true`.
AddTerminationCondition("function") Add a boolean `function`, episode will end if `true`.
StateSpaceDefine("function", "name") A function that takes the raw state space as input, and places the output into a new state vector `name`.
SourcePolicy("sourcepolicy","iscopy") Policy currently being learned should be based on `sourcepolicy`.
AddAdvicePolicy("policy") Adds a policy giving advice.
AdviseAction("state", "vals", "function") When `state` equals `vals`, recommend the action in `function`.
StartLearning("until") Starts learning `until` a condition (e.g., `number_of_timesteps = k`) is reached.

3. TEACHING ROBOCUP KEEPAWAY

We have implemented the language and tested it in a simple grid world task, a more complex Blocks World task, and the RoboCup keepaway benchmark task. In this abstract we focus on Robocup soccer, a fully distributed, multiagent domain with both teammates and adversaries. *Keepaway* is a subtask of RoboCup, 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[1] . Whenever the takers take possession or the ball

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

```
1  BeginTaskDescription("BallHandlingPolicy");
2  Environment("RoboCupSoccerSimulator");
3  BeginEpisode( "InitializeKeepaway(3, 2, 25, 25)" );
4  WrapperPolicy("KeeperPolicy", "BallIsKickable" )
5  AddToActionSpace( "HoldBall()" );
6  AddToActionSpace( "PassKThenReceive()" );
7  AddAdviceActions( "HandCodedBallHandling" );
8  StateSpaceDefs( "SortKeeperDistances", "K");
9  StateSpaceDefs( "SortTakerDistances", "T" );
10 AddToStateSpace( "dist(K[1], C)" );
11 AddToStateSpace( "dist(K[2], C)" );
12 AddToStateSpace( "dist(K[3], C)" );
13 AddToStateSpace( "dist(T[1], C)" );
14 AddToStateSpace( "dist(T[2], C)" );
15 AddToStateSpace( "dist(K[1], K[2])" );
16 AddToStateSpace( "dist(K[1], K[3])" );
17 AddToStateSpace( "dist(K[1], T[1])" );
18 AddToStateSpace( "dist(K[1], T[2])" );
19 AddToStateSpace( "Min(dist(K[2], T[1]), dist(K[2], T[2]))");
20 AddToStateSpace( "Min(dist(K[3], T[1]), dist(K[3], T[2]))");
21 AddToStateSpace( "Min(ang(K[2], K[1], T[1]),
22                    ang(K[2], K[1], T[2]))" );
23 AddToStateSpace( "Min(ang(K[3], K[1], T[1]),
24                    ang(K[3], K[1], T[2]))" );
25 AddTerminationCondition( "TakerHoldsBall" );
26 AddTerminationCondition( "BallOutOfBounds" );
27 StepReward( "TimeElapsed(now, lastStepStart)" );
28 OnEpisodeEnd( "RestartFromBeginning" );
29 StartLearning();
```

Figure 1: BLTL instructions for the Keepaway ball-handling task.

leaves the region, the episode ends and the players are reset for another episode. The goal in this paper is to learn the *ball handling* policy, in which the keeper with the ball must decide at every timestep whether to hold the ball or to kick to one of its teammates (and to which one). For the agent to learn an effective policy, the teacher must first specify how to start and end an episode, then tell the student what the state and action spaces are, then allow the student to practice the game over multiple episodes.

Figure 1 shows the series of instructions needed to specify the keepaway ball-handling task. The 13 state variables when the keeper has the ball, defined by the series of AddToStateSpace() functions, are the same as the ones commonly used for learning this task [11], and consist of several distances and angles among the keepers and takers. Note that the functions map relatively easily from natural language to the formal specification. For example, "learn a new soccer-related task" is represented in line 2; "base your decision in part on the distances of the players to the center of the field" is represented in lines 10–24; and "Use the existing hand-coded policy as advice" on line 7. Such NLP in a constrained domain is currently possible [5] and would enable a domain expert with no special RL knowledge to express the learning task to the BL student.

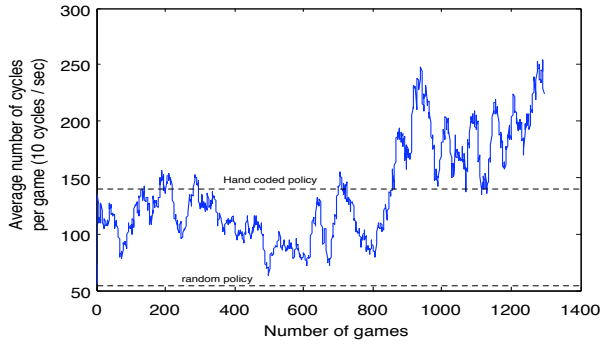

Figure 2: Performance results with Keepaway using advice.

3.1 Keepaway Results

We were able to initiate learning for keepaway identically to the description in [11], and then we also added a hand-coded policy as a form of advice, allowing the learning agent to implement an advice-taking algorithm similar to that of [5]. Figure 2 shows average game times as learning progresses. Each point on the curve represents the average time of the previous 50 games. The agents quickly achieve good average game times of about 15 seconds (slightly better than the hand-coded policy that is being used for giving advice), and eventually achieve average game times of up to 25 seconds, about equal with the best reported results on keepaway from [4] but with far less training.

4. FUTURE WORK

An important next step is to test the BLTL language for other tasks, both in RoboCup and in completely different domains such as flying an unmanned aerial vehicle (UAV). The full Bootstrap Learning framework will include natural language mapping to the BLTL language, and will add more machine learning methods. The BLTL language also lays the groundwork for future development of agents which can decide for themselves what tasks to learn and how to learn them, rather than waiting for task specifications and advice from a teacher.

5. REFERENCES

[1] Bootstrapped learning proposer information pamphlet. `http://www.darpa.mil/IPTO/solicit/closed/BAA-07-04_PIP.pdf`.

[2] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *NIPS 2005*. Morgan Kaufmann, San Mateo, CA, 1993.

[3] T. G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *ICML*. Madison, WI, 1998.

[4] T. Jung and D. Polani. Learning robocup-keepaway with kernels. In *JMLR Workshop and Conference Proceedings ("Gaussian Processes in Practice")*, volume 1. JMLR, 2007.

[5] G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in robocup soccer. In *AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems*, 2004.

[6] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

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

[8] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[9] S. Schaal. Learning from demonstration. *NIPS 9*, 1997.

[10] V. Soni and S. Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of the Twenty First National Conference on Artificial Intelligence*, July 2006.

[11] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[12] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112, 1999.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[14] M. E. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In *AAMAS 2005*, pages 53–59, July 2005.

[15] L. Torrey, T. Walker, J. Shavlik, and R. Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *ECML 2005*. Porto, Portugal, 2005.

[16] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[17] A. White, M. Lee, A. Butcher, B. Tanner, L. Hackman, and R. Sutton. Rl-glue distribution, *http://rlai.cs.ualberta.ca/rlbb/top.html*, 2007.