In Baltes, Lagoudakis, Naruse, Gidary, editors, *RoboCup 2009: Robot Soccer World Cup XIII*, pp. 153--165, Springer, 2010.

Learning Complementary Multiagent Behaviors: A Case Study

Shivaram Kalyanakrishnan and Peter Stone
Department of Computer Sciences, The University of Texas at Austin
{shivaram, pstone}@cs.utexas.edu

Abstract. As machine learning is applied to increasingly complex tasks, it is likely that the diverse challenges encountered can only be addressed by combining the strengths of different learning algorithms. We examine this aspect of learning through a case study grounded in the robot soccer context. The task we consider is Keepaway, a popular benchmark for multiagent reinforcement learning from the simulation soccer domain. Whereas previous successful results in Keepaway have limited learning to an isolated, infrequent decision that amounts to a turn-taking behavior (passing), we expand the agents' learning capability to include a much more ubiquitous action (moving without the ball, or getting open), such that at any given time, multiple agents are executing learned behaviors simultaneously. We introduce a policy search method for learning "GetOpen" to complement the temporal difference learning approach employed for learning "Pass". Empirical results indicate that the learned GetOpen policy matches the best hand-coded policy for this task, and outperforms the best policy found when Pass is learned. We demonstrate that Pass and GetOpen can be learned simultaneously to realize tightly-coupled soccer team behavior.

1 Introduction

Learning to play soccer can be framed elegantly as a multiagent reinforcement learning (RL) problem. However, the state-of-the-art in multiagent RL is yet to cope with the demands of such a complex problem. In the context of multiagent RL, a number of models have been proposed to exploit task-specific regularities such as coordination of actions [5], state abstraction [4], and information sharing [12]. While such measures all pave the way towards learning increasingly complex behavior, they still assume that the task being considered is simple enough to be learned using a *single* learning algorithm. Yet complex tasks such as soccer comprise multiple overlapping behaviors, whose diverse demands can only be met by combining the strengths of qualitatively different learning approaches. Identifying this as a crucial direction for future research, we present a detailed case study of one such task that is grounded in the RoboCup 2D simulation soccer platform [2].

The task we consider is Keepaway [14], which has become a popular test-bed for multiagent RL [8, 9]. Keepaway is a realistic, continuous, high-dimensional, stochastic task, and is significantly more complex than synthetic, discrete tasks such as Predator-Prey [1] that have been used in the past for studying agent cooperation [7] and games such as Tic-Tac-Toe for studying agent competition [12]. However, all the learning in Keepaway to date has addressed just one aspect of the task, in which the learned decision is made on a turn-taking basis among teammates. These studies have all focused on the "Pass" behavior of the player with possession of the ball in deciding whether (and to which teammate) to pass. They assume that its teammates, when moving to positions on the field likely to induce successful passes, execute fixed, hand-coded "GetOpen" strategies.

In contrast, we formulate GetOpen as a multiagent learning problem, thereby extending learning in Keepaway from Pass to Pass+GetOpen. Consequently, Keepaway becomes an instance of a learning problem composed of highly interdependent behaviors executing simultaneously. Each player executes multiple behaviors (Pass and GetOpen) that affect the outcome of its teammates' behaviors, and in the long run, also interact with one another. Such a scenario poses a significant challenge for designing a credit assignment scheme that both reflects the intended objectives in the underlying task and guides learning in a natural, incremental manner.

We present a novel solution for learning GetOpen using policy search, which contrasts with the temporal difference learning method used for Pass. Results show that the learned GetOpen policy matches the best performing hand-coded policy for this task. Further experiments illustrate that learning these complementary behaviors results in a tight coupling between them, and indeed that Pass and GetOpen can be learned simultaneously. These results demonstrate the effectiveness of applying separate learning algorithms to distinct components of a significantly complex task. As a direct consequence of our formulation of GetOpen for learning, numerous opportunities arise for conducting research in the Keepaway test-bed.

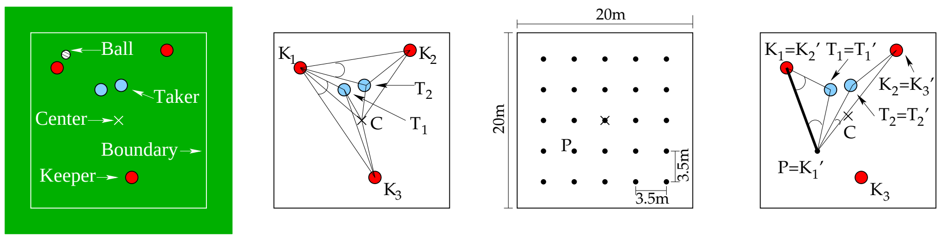
This paper is organized as follows. In Section 2, we review the standard Pass task and formalize GetOpen similarly. In Section 3, we describe algorithms for learning Pass and GetOpen, both individually and together. Experimental results are discussed in Section 4, which is followed by a presentation of related and future work in Section 5. Our conclusions are summarized in Section 6.

2 Keepaway Pass and GetOpen

The RoboCup 2D simulation soccer domain [2] models several difficulties that agents must cope with in the real world. Soccer is necessarily a multiagent enterprise, in which agents have both teammates and opponents. In the simulation, they are only provided partial and noisy perceptions, and have imperfect actuators. Their sensing and acting routines are not synchronized, and in the interest of keeping real time, do not admit extensive deliberation. The atomic actions available to an agent are Turn, Turn-Neck, Dash, Kick, and Catch; skills such as passing to a teammate or going to a point must be composed of a string of these low-level actions executed sequentially. For all these reasons, simulated RoboCup soccer becomes a challenging domain for machine learning.

Keepaway [14] is a subtask of soccer in which a team of 3 *keepers* aims to keep possession of the ball[1] away from the opposing team of 2 *takers*. The game

[1] A player is deemed to have *possession* of the ball if it is close enough to be kicked.



(a) (b) (c) (d)

Fig. 1. (a) A snapshot of Keepaway. (b) Corresponding Pass state variables. (c) Target points for GetOpen, among them P. (d) Corresponding GetOpen state variables. $dist(K_1', K_2')$ and $dist(K_1, K_1')$ (darkened) overlap.

is played within a square region of side $20m$.[2] Each episode begins with some keeper having the ball, and ends when some taker claims possession or the ball overshoots the region of play. It is the objective of the keepers to maximize the expected length of the episode, referred to as the episodic **hold time**. The keepers must cooperate with each other in order to realize this objective; they compete with the team of takers that seeks to minimize the hold time. Figure 1(a) shows a snapshot of a Keepaway episode in progress.

In order to make the task amenable to learning, it becomes necessary to constrain the scope of decision making by the keepers. Figure 2 outlines the policy followed by *each* keeper in the scheme employed by Stone *et al.* [14]. The keeper closest to the ball intercepts the ball until it has possession. Once it has possession, it must execute the Pass behavior (not to be confused with a pass action), by way of which it may retain ball possession or pass to a teammate. Keepers other than the one closest to the ball move to a position conducive for receiving a pass by executing GetOpen behavior.

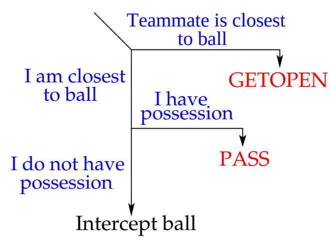


Fig. 2. Policy followed by each keeper.

Pass and GetOpen, by offering a *choice* of high-level actions based on the keeper's state, are candidates for the application of learning. Most prior work assumes GetOpen, and indeed the behavior followed by the takers, to follow fixed, hand-coded strategies. In other words, the teammates and opponents of the keeper with the ball do not *adapt* to the specific characteristics of that keeper, as they do in real soccer. As a step in the direction of furthering team adaptation, we extend the frontier of learning in Keepaway to include GetOpen. Thus, we treat Keepaway as a composite of two distinct behaviors to be learned: Pass and GetOpen. As in previous work [14], we restrict the takers to the fixed policy of moving towards the ball. In recent work, Iscen and Erogul [8] explore learning taker behavior, which complements the work in our paper (see Section 5).

[2] Keepaway can be generalized to varying numbers of keepers and takers, as well as field sizes [14].

2.1 Keepaway PASS

Here we revisit the problem of Pass defined by Stone *et al.* [14]. The keepers and takers assume roles that are indexed based on their distances to the ball: K_i is the i^{th} closest keeper to the ball, and T_j the j^{th} closest taker. From Figure 2, we see that the keeper executing Pass must be K_1.

The three high-level actions available to K_1 are HoldBall, which is composed of a series of kicks close to itself, but away from any approaching takers; and PassBall-i, $i \in 2, 3$, a direct pass to K_i. Each player processes its low-level perceptual information to construct a world model, which constitutes a continuous state space. This space is represented through a vector of 13 state variables, comprising distances and angles among the players and the center C of the field. These are marked in Figure 1(b), and enumerated in Table 1.

A policy for Pass maps a 13-dimensional vector representing the state variables to one of the high-level actions: HoldBall, PassBall-2, and PassBall-3. An example of such a policy is Pass:Hand-coded (Algorithm 1), which implements a well-tuned manually programmed strategy [14]. Under this policy, K_1 executes HoldBall until the takers get within a certain range, after which distances and angles involving its teammates and opponents are used to decide whether (and to which teammate) to pass. Yet another policy for Pass is Pass:Random, under which K_1 chooses one of the three available actions with equal likelihood. Pass:Learned denotes a learned Pass policy, which is described in Section 3.

Algorithm 1 Pass:Hand-coded

input Pass state variables (13)
output Action $\in$ {HoldBall, PassBall-2, PassBall-3}
  if $dist(K_1, T_1) > C_1$ **then**
    Return HoldBall.
  for $i \in 2, 3$ **do**
    $valAng_i \leftarrow \min_{j \in 1,2} ang(K_i, K_1, T_j)$.
    $valDist_i \leftarrow \min_{j \in 1,2} dist(K_i, T_j)$.
    $val_i \leftarrow C_2 \cdot valAng_i + valDist_i$.
  if $\max_{i \in 2,3} val_i > C_3$ **then**
    $passIndex \leftarrow \text{argmax}_{i \in 2,3}\, val_i$.
    Return PassBall-passIndex.
  else
    Return HoldBall.
{$C_1 = 5.0, C_2 = 0.25, C_3 = 22.5$; distances are taken to be in meters and angles in degrees.}

2.2 Keepaway GETOPEN

Whereas learning the Pass behavior has been studied extensively in the literature [9, 10], to the best of our knowledge, all previous work has used the hand-coded GetOpen policy originally defined by Stone *et al.* [14], which we refer to here as GetOpen:Hand-coded. Thus, while previous work on this task has considered multiple agents learning, they have never been executing their learned behaviors concurrently (only one player executes Pass at any given time). This paper introduces a learned GetOpen behavior, thereby expanding the scope of multiagent learning in Keepaway significantly. Below we describe our formulation of GetOpen.

In principle, there are infinitely many positions that K_2 and K_3 can occupy on the square playing field, However, they only get a small amount of time to pick a target. Since nearby points are likely to be of similar value, an effective strategy is to evaluate only a small, finite set of points spread out across the field and choose the most promising. Figure 1(c) shows a uniform grid of 25 points overlaid on the field, with a 15% margin on the sides. GetOpen is implemented by evaluating each grid point P, and moving to the one with the highest value. Indeed, we define the GetOpen learning problem to be learning an evaluation function that assigns a value to every target point P, given the configuration of the players.

As with Pass, it becomes necessary to define a set of state variables for learning GetOpen. In Figure 1(d), K_3 is shown seeking to evaluate the point P at some time t. The distances and angles marked correspond to the GetOpen state variables used for the purpose, which we identify based on informal experimentation. None of the state variables involve K_3, as K_3 is examining a situation at time t' in the future when it would itself be at P. At time t', K_3 expects to have possession of the ball, and re-orders the other players based on their distances to it. Thus K_3 becomes K'_1, and in the state from Figure 1(d), K_1 becomes K'_2, T_1 becomes T'_1, and so on. Conceptually, the evaluation of the target point P should consider both the likelihood of receiving a pass at P, and the value of being at P with the ball afterwards. This leads to two logical groups within the state variables. One group contains 2 variables that influence the success of a pass from K_1 to K'_1, the latter being at P. These are the distance between K_1 and K'_1, and the minimum angle between K_1, K'_1 and any taker. The other group of state variables bear direct correspondences with those used for learning Pass, but computed under the re-ordering at t'. Of the 13 state variables used for Pass, we leave out the 5 distances between the players and the center of the field, as they do not seem to benefit the learning of GetOpen. This results in a total of 10 state variables for GetOpen, which are listed in Table 1.

Table 1. Pass, GetOpen state variables.

Pass	GetOpen
$dist(K_1, K_2)$	$dist(K'_1, K'_2)$
$dist(K_1, K_3)$	$dist(K'_1, K'_3)$
$dist(K_1, T_1)$	$dist(K'_1, T'_1)$
$dist(K_2, T_2)$	$dist(K'_2, T'_2)$
$\min_{j \in 1,2} dist(K_2, T_j)$	$\min_{j \in 1,2} dist(K'_2, T'_j)$
$\min_{j \in 1,2} ang(K_2, K_1, T_j)$	$\min_{j \in 1,2} ang(K'_2, K'_1, T'_j)$
$\min_{j \in 1,2} dist(K_3, T_j)$	$\min_{j \in 1,2} dist(K'_3, T'_j)$
$\min_{j \in 1,2} ang(K_3, K_1, T_j)$	$\min_{j \in 1,2} ang(K'_3, K'_1, T'_j)$
$dist(K_1, C)$	$dist(K_1, K'_1)$
$dist(K_2, C)$	$\min_{j \in 1,2} ang(K'_1, K_1, T_j)$
$dist(K_3, C)$	
$dist(T_1, C)$	
$dist(T_2, C)$	

In defining the state variables for GetOpen, it is implicitly assumed that players other than K'_1 do not change their positions between t and t'. This clearly imperfect assumption does not have too adverse an impact since GetOpen is executed *every* cycle, always with the *current* positions of all players. Revising the target point every cycle, however, has an interesting effect on a random GetOpen policy. In order to get from point A to point B, a player must first turn towards B, which takes 1-2 cycles. When a random target point is chosen each cycle, K'_1 constantly keeps turning, achieving little or no net displacement. To redress this effect, our implementation of GetOpen:Random only allows K'_1 to revise its target point when it reaches its current target. Such a measure is not necessary when the targets remain reasonably stable, as they do for GetOpen:Learned, the learned policy, and GetOpen:Hand-coded [14], which we describe below.

Under GetOpen:Hand-coded (Algorithm 2), the value of a point P is inversely related to its *congestion*, a measure of its distances to the keepers and takers. Assuming that K_1 will pass the ball from *predictedBallPos*, P is deemed an inadmissible target (given a value of $-\infty$) if any taker comes within a threshold angle of the line joining *predictedBallPos* and P. Thus, GetOpen:Hand-coded is a sophisticated policy using complex entities such as congestion and the ball's predicted position, which are not captured by the set of state variables we define for learning GetOpen. In Section 4, we compare GetOpen:Hand-coded with GetOpen:Learned to verify if simple distances and angles indeed suffice for describing competent GetOpen behavior.

Algorithm 2 GetOpen:Hand-coded

input Evaluation point P, World State
output Value at P
$teamCongestion \leftarrow \sum_{i \in 1,2,3, i \neq myIndex} \frac{1}{dist(K_i, P)}$.
$oppCongestion \leftarrow \sum_{j \in 1,2} \frac{1}{dist(T_j, P)}$.
$congestion \leftarrow teamCongestion + oppCongestion$.
$value \leftarrow -congestion$.
$safety \leftarrow \min_{j \in 1,2} ang(P, predictedBallPos, T_j)$.
if $safety < C_1$ **then**
$value \leftarrow -\infty$.
Return *value*.
$\{C_1 = 18.4$; angles are taken to be in degrees.$\}$

2.3 Keepaway PASS+GETOPEN

Pass and GetOpen are separate behaviors of the keepers, which together may be viewed as "distinct populations with coupled fitness landscapes" [12]. At any instant, there are two keepers executing GetOpen; their teammate, if it has intercepted the ball, executes Pass. Specifically, each keeper executes GetOpen when it assumes the role of K_2 or K_3, and executes Pass when it has possession of the ball, as K_1. The extended sequence of actions that results as a combination each keeper's Pass and GetOpen policies determines the team's performance. Indeed, the episodic hold time is precisely the temporal length of that sequence. Pass has been the subject of many previous studies, in which it is modeled as a (semi) Markov Decision Problem (MDP) and solved through temporal difference learning (TD learning) [9, 10, 14]. In Pass, each action (HoldBall, PassBall-1, PassBall-2) is taken by exactly one keeper; hence only the keeper that takes an action needs to get rewarded for it. Indeed, if this reward is the time elapsed until the keeper takes its next action (or the episode ends), the episodic hold time gets maximized if each keeper maximizes its own long-term reward.

Unfortunately, GetOpen does not admit a similar credit assignment scheme, because at any instant, two keepers (K_2 and K_3) take GetOpen actions to move to target points. If K_1 executes the HoldBall action, none of them will reeceive a pass; if K_1 passes to K_2 (K_3), it is not clear how K_3 (K_2) should be rewarded. In principle, the sequence of *joint actions* taken by K_2 and K_3 up to the successful pass must be rewarded. Yet, such a joint action is taken *every* cycle (in contrast with Pass actions, which last 4-5 cycles on average), and the large number of atomic GetOpen actions (25, compared to 3 for Pass) leads to a very large joint action space. In short, GetOpen induces a far more complex MDP than Pass. An additional obstacle to be surmounted while learning Pass and GetOpen together is non-stationarity introduced by each into the other's environment. All these reasons, combined with the inherent complexity of RoboCup 2D simulation soccer, make Pass+GetOpen a demanding problem for machine learning.

3 Learning Framework

Each of the 3 keepers must learn one Pass and one Get-Open policy; an array of choices exists in deciding whether the keepers learn separate policies or learn them in common. Thus, the total number of policies learned may range from 2 (1 Pass, 1 GetOpen) to 6 (3 Pass, 3 GetOpen). Different configurations have different advantages in terms of the size of the overall search space, constraints for communication, the ability to learn specialized behaviors, etc. It falls beyond the scope of this paper to systematically comb the space of solutions for learning Pass and GetOpen. As an exploratory study, our emphasis in this work is rather on verifying the *feasibility* of learning these behaviors, guided by intuition, trial and error. In the learning scheme we adopt, each keeper learns a unique Pass policy, while all of them share a common GetOpen policy. We proceed to describe these. As in Section 2, we furnish pseudo-code and parameter settings to ensure that our presentation is complete and our experiments reproducible.

3.1 Learning PASS

We apply the *same* algorithm and parameter values employed by Stone *et al.* for learning Pass [14], under which each keeper uses Sarsa to make TD learning updates. Owing to space restrictions, we do not repeat the specifications of this method here, which is described in detail in Section 4 of their paper [14].

3.2 Learning GETOPEN

The solution to be learned under GetOpen is an evaluation function over its 10 state variables, by applying which the keepers maximize the hold time of the episode. Whereas TD learning is a natural choice for learning Pass, the difficulties outlined in Section 2.3 to solve GetOpen as a sequential decision making problem make direct policy search a more promising alternative. Thus, we represent the evaluation function as a parameterized function and search for parameter values that lead to the highest episodic hold time.

Our learned GetOpen policy is implicitly represented through a neural network that computes a value for a target location given the 10-dimensional input state. The player executing GetOpen compares the values at different target points on the field, and moves to the point with the highest value. Note that unlike with Pass, these values do not have the same semantics as action values computed through TD learning; rather, they merely serve as *action preferences*, whose relative order determines which action is chosen. We achieve the best results using a 10-5-5-1 network, with a total of 91 parameters (including biases at each hidden node). The parameters are initialized to random values drawn uniformly from $[-0.5, 0.5]$; each hidden node implements the sigmoid function $f(x) = 1.7159 \cdot tanh(\frac{2}{3}x)$, suggested by Haykin [6].

A variety of policy search methods are applicable for optimizing the 91-dimensional policy. We verify informally that methods such as hill climbing, genetic algorithms, and policy gradient methods all achieve qualitatively similar results. The experiments reported in this paper are conducted using the cross-entropy method [3], which evaluates a population of candidate solutions drawn from a distribution, and progressively refines the distribution based on a selection the fittest candidates. We use a population size of 20 drawn initially from $N(0,1)^{91}$, picking the fittest 5 after each evaluation of the population. Each keeper follows a fixed, stationary PASS policy across all evaluations in a generation; within each evaluation, all keepers share the same GETOPEN policy (the one being evaluated). The fitness function used is the average hold time over 125 episodes, which negates the high stochasticity of Keepaway.

3.3 Learning PASS+GETOPEN

Algorithm 3 outlines our method for learning PASS+GETOPEN. Learning is bootstrapped by optimizing a GETOPEN policy for a random PASS policy. The best GETOPEN policy found after two iterations (a total of $2 \times 20 \times 125 = 5000$ episodes) is fixed, and followed while learning PASS using Sarsa for the next 5000 episodes. The PASS policy is now frozen, and GETOPEN is once again improved. Thus, inside the outermost loop, either PASS or GETOPEN is fixed and stationary, while the other is improved, starting from its current value. Note that π_{Pass} and π_{GetOpen} are still *executed* concurrently during each Keepaway episode as part of *learnPass*() and *learnGetOpen*().

Algorithm 3 Learning PASS+GETOPEN

output Policies π_{Pass} and π_{GetOpen}
  $\pi_{\text{Pass}} \leftarrow$ PASS:RANDOM.
  $\pi_{\text{GetOpen}} \leftarrow$ GETOPEN:RANDOM.
  repeat
    $\pi_{\text{GetOpen}} \leftarrow learnGetOpen(\pi_{\text{Pass}}, \pi_{\text{GetOpen}})$.
    $\pi_{\text{Pass}} \leftarrow learnPass(\pi_{\text{Pass}}, \pi_{\text{GetOpen}})$.
  until convergence
  Return $\pi_{\text{Pass}}, \pi_{\text{GetOpen}}$.

Whereas Algorithm 3 describes a general learning routine for each keeper to follow, in our specific implementation, the keepers execute it in phase, and indeed share the same π_{GetOpen}. Also, we obtain slightly better performance in learning PASS+GETOPEN by spending more episodes on learning GETOPEN than on learning PASS, which we report in the next section.

4 Results and Discussion

In this section, we report the results of a systematic study pairing three PASS policies (PASS:RANDOM, PASS:HAND-CODED, and PASS:LEARNED) with three GETOPEN policies (GETOPEN:RANDOM, GETOPEN:HAND-CODED, and GETOPEN:LEARNED). For the sake of notational convenience, we use abbreviations: thus, PASS:RANDOM is denoted P:R, GETOPEN:LEARNED is denoted GO:L, and their conjunction P:R-GO:L. Nine configurations arise in total. Figure 4 shows the performance of each PASS policy when paired with different GETOPEN policies, and vice versa.[3] Policies in which both PASS and GETOPEN are either random or hand-coded are static, while the others show learning.

Figure 3(c) shows the performance of P:L. P:L-GO:HC corresponds to the experiment conducted by Stone *et al.* [14], and we see similar results. After 30,000 episodes of training, the hold time achieved is about 14.9 seconds, which falls well short of the 16.7 seconds registered by the static P:HC-GO:HC policy (Figure 3(b)). Although P:L-GO:HC is trained in these experiments with a

[3] Videos of policies are posted on the following web page: `http://www.cs.utexas.edu/~AustinVilla/sim/keepaway-getopen/`.

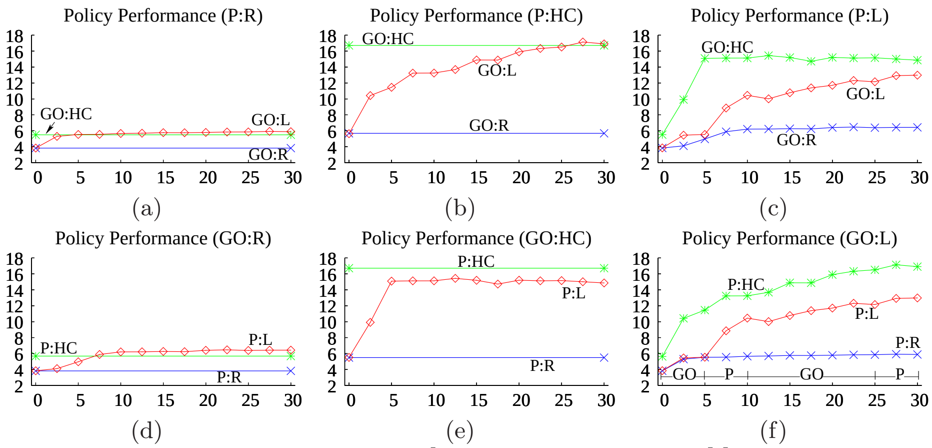


Fig. 3. Learning curves corresponding to conjunctions of various PASS and GETOPEN policies. Each curve represents an average over at least 20 independent trials. Each reported point corresponds to an evaluation (non-learning) for 500 episodes; points are reported every 2500 episodes. Note that each of the nine experiments appears once in the left column, where experiments are grouped by common PASS policies, and once in the right column, where they are grouped by GETOPEN.

constant learning rate of $\alpha = 0.125$, we posit that annealing α will improve its performance by avoiding the gradual dip in hold time we observe between episodes 12,500 and 30,000. In the absence of any guarantees about convergence to optimality, we consider the well-tuned P:HC-GO:HC to serve as a near-optimal benchmark for the learning methods. Interestingly, under the random GETOPEN policy GO:R (Figure 3(d)), P:HC is overtaken by P:L at 30,000 episodes ($p < 0.0001$). This highlights the ability of learning methods to adapt to different settings, for which hand-coded approaches demand manual attention.

Figure 3(f) confirms the viability of our policy search method for learning GETOPEN, and its robustness in adapting to different PASS policies. Practical considerations force us to terminate experiments after 30,000 episodes of learning, which corresponds roughly to one day of real training time. After 30,000 episodes, P:HC-GO:L achieves a hold time of 16.9 seconds, which indeed exceeds the hold time of P:HC-GO:HC (Figure 3(b)); yet despite running 20 independent trials of each, this result is not statistically significant. Thus, we only conclude that when coupled with P:HC, learning GETOPEN, a novel contribution of this work, matches the hand-coded GETOPEN policy that has been used in all previous studies on the Keepaway task. This result also highlights that well-crafted state variables such as *congestion* and *predictedBallPos*, which are used by P:HC-GO:HC, are not necessary for describing good GETOPEN behavior. Interestingly, the hold time of P:HC-GO:L is significantly higher than that of P:L-GO:HC ($p < 0.001$). In other words, our GETOPEN learning approach outperforms the previously studied PASS learning when each is paired with a hand-coded counterpart, underscoring the relevance of *learning* GETOPEN.

An important result we observe from Figures 3(c) and 3(f) is that not only can Pass and GetOpen be learned when paired with static policies, they can indeed be learned in tandem. In our implementation of Algorithm 3, we achieve the best results by first learning GetOpen using policy search for 5000 episodes, followed by 5000 episodes of learning Pass using Sarsa. Subsequently, we conduct 6 generations of learning GetOpen (episodes 10,000 to 25,000), followed by another 5000 episodes of Sarsa, as depicted along the x axis in Figure 3(f). The hold time of P-L:GO-L (13.0 seconds after 30,000 episodes) is significantly lower than P:L-GO:HC, P:HC-GO:L, and P:HC-GO:HC ($p < 0.001$), reflecting the additional challenges encountered while learning Pass and GetOpen simultaneously. Indeed, we notice several *negative* results with other variant methods for learning Pass+GetOpen. In one approach, we represent both Pass and GetOpen as parameterized policies and evolve their weights concurrently to maximize hold time. In another approach, GetOpen uses the value function being learned by Pass as the evaluation function for target points. In both these cases, the performance never rises significantly above random.

We conduct a further experiment in order to ascertain the degree of specialization achieved by learned Pass and GetOpen policies, i.e., whether it is beneficial to learn Pass specifically for a given GetOpen policy (and vice versa). In Table 2, we summarize the performances of learned Pass and GetOpen policies trained and tested with different counterparts. Each column corresponds to a test pairing. We notice that the best performing Pass policy for a given GetOpen policy is one that was trained with the same GetOpen policy (and vice versa); the maximal sample mean in each column coincides with the diagonal. It must be noted, however, that despite conducting at least 20 trials of each experiment, some comparisons are not statistically significant. A possible reason for this is the high variance caused by the stochasticity of the domain. Yet, it is predominantly the case that learned behaviors adapt to work best with the counterpart behavior with which they are playing. Thus, although different learning algorithms are applied to Pass and GetOpen, the behaviors are tightly-coupled in the composite solution learned.

Table 2. In the table on the left, Pass learned while trained with different GetOpen policies is tested against different GetOpen policies. Each entry shows the mean hold time and one standard error of at least 20 independent runs, conducted for 500 episodes. Each *column* corresponds to a test GetOpen policy. The largest entry in each column is in boldface; entries in the same column are marked with "-" if not significantly lower ($p < 0.05$). The cell GO:L-GO:L shows two entries: when the learned Pass policy is tested against the same ("s") learned GetOpen policy as used in training, and when tested against a different ("d") learned GetOpen policy. The table on the right is constructed similarly for GetOpen, and uses the same experiments as Pass for the cell P:L-P:L.

Pass:Learned			
Train	Test		
	GO:R	GO:HC	GO:L
GO:R	**6.37**±.05	11.73±.25	10.54±.26
GO:HC	6.34±.06$^-$	**15.27**±.26	12.25±.32
GO:L	5.96±.07	13.39±.35	**13.08**±.26 (s)
			12.32±.32 (d)$^-$

GetOpen:Learned			
Train	Test		
	P:R	P:HC	P:L
P:R	**5.89**±.05	10.40±.39	11.15±.43
P:HC	5.48±.04	**16.89**±.39	12.99±.43$^-$
P:L	5.57±.06	11.78±.56	**13.08**±.26 (s)
			12.32±.32 (d)$^-$

5 Related and Future Work

Multiple learning methods are used in the layered learning architecture developed by Stone [13] for simulated soccer. These include neural networks for learning to intercept the ball, decision trees for evaluating passes, and TPOT-RL, a TD learning method for high-level strategy learning. This work shares our motivation that different sub-problems in a complex multiagent learning problem can benefit from specialized solutions. Yet a key difference is that in Stone's architecture, skills learned using supervised learning are employed in higher-level sequential decision making, to which RL is applied; in our work, the two learning problems we consider are themselves both sequential decision making problems.

The policy search approach we use for GetOpen is similar to one used by Haynes *et al.* [7] for evolving cooperative behavior among four predators that must collude to catch a prey. The predators share a common policy, represented as a LISP S-expression, in contrast with the neural representation we engage for computing a real-valued evaluation function. The Predator-Prey domain [1], which is discrete and non-stochastic, is much simpler compared to Keepaway.

By decomposing Keepaway into Pass and Getopen, our work enriches the multiagent nature of the problem and spawns numerous avenues for future work. For example, a new promising dimension is agent communication. Consider K_1 "yelling" to K_2 where it is about to pass, as is common in real soccer. K_1's Pass and K_2's GetOpen behaviors could conceivably exploit such information to further team performance.

The Brainstormers team [11] has applied RL for learning attacking team behavior. In their work, the actions available to the player with the ball are several variants of passing and dribbling. Its teammates can move in different directions or head to a home position. Assuming the availability of an environmental model, TD learning is used to estimate a value function over the possible states. The team attack is shown to increase its goal-scoring percentage. Iscen and Erogul [8] consider applying TD learning to the behavior of the takers. The actions available to the takers are ball interception and player marking. Whereas Pass+GetOpen models cooperation, extending Keepaway to include taker behavior would also incorporate competition.

6 Conclusion

Through a concrete case study, we advance the case for applying different learning algorithms to qualitatively distinct behaviors present in a complex multiagent system. In particular, we introduce Keepaway GetOpen as a multiagent learning problem that complements Keepaway Pass, the well-studied reinforcement learning test-bed problem from the robot soccer domain. We provide a policy search method for learning GetOpen, which compares on par with a well-tuned hand-coded GetOpen policy, and which can also be learned simultaneously with Pass to realize tightly-coupled behaviors. Learning GetOpen with a hand-coded Pass policy outperforms the earlier result in which Pass is learned and GetOpen is hand-coded. Our algorithm for learning both Pass and GetOpen in an interleaved manner confirms the feasibility of learning them together, but also shows significant scope for improvement. This work widens the scope for conducting research on the Keepaway test-bed. It puts together distinct techniques that apply to sequential decision making, which is a crucial element in scaling to more complex multiagent learning problems.

Acknowledgments

The authors thank Ian Fasel and anonymous reviewers of the current and previous versions of this paper for providing useful comments. This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (CNS-0615104, EIA-0303609 and IIS-0237699), DARPA (FA8750-05-2-0283, FA-8650-08-C-7812 and HR0011-04-1-0035), General Motors, and the Federal Highway Administration (DTFH61-07-H-00030).

References

1. M. Benda, V. Jagannathan, and R. Dodhiawala. On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Comp. Serv., Seattle, WA, July 1986.
2. M. Chen, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostiadis, J. Kummeneje, I. Noda, O. Obst, P. Riley, T. Steffens, Y. Wang, and X. Yin. Users manual: RoboCup soccer server — for soccer server version 7.07 and later. *The RoboCup Federation*, August 2002.
3. P. T. De Boer, D. P. Kroese, S. Mannor, and R. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.
4. M. Ghavamzadeh, S. Mahadevan, and R. Makar. Hierarchical multi-agent reinforcement learning. *Aut. Agents and Multi-Agent Sys.*, 13(2):197–229, 2006.
5. C. Guestrin, M. G. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In C. Sammut and A. G. Hoffmann, editors, *Proceedings of the Nineteenth International Conference on Machine Learning, University of New South Wales, Sydney, Australia, July 8-12, 2002*, pages 227–234. Morgan Kaufmann, 2002.
6. S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.
7. T. Haynes, R. Wainwright, S. Sen, and D. Schoenefeld. Strongly typed genetic programming in evolving cooperation strategies. In S. Forrest, editor, *Proc. of the 6^{th} Int. Conf. Gen. Alg.*, pages 271–278, San Mateo,CA, 1995. Morgan Kaufman.
8. A. Iscen and U. Erogul. A new perpective to the keepaway soccer: The takers. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi–Agent Systems*, pages 1341–1344, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.
9. T. Jung and D. Polani. Learning Robocup-Keepaway with kernels. *Gaussian Processes in Practice: JMLR Workshop and Conference Proceedings*, 1:33–57, 2007.
10. J. H. Metzen, M. Edgington, Y. Kassahun, and F. Kirchner. Analysis of an evolutionary reinforcement learning method in a multiagent domain. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi–Agent Systems*, pages 291–298, Richland, SC, May 2008. International Foundation for Autonomous Agents and Multiagent Systems.
11. M. Riedmiller and T. Gabel. On experiences in a complex and competitive gaming domain: Reinforcement learning meets robocup. *3rd IEEE Symposium on Computational Intelligence and Games*, pages 17–23, April 2007.
12. C. D. Rosin and R. K. Belew. Methods for competitive co-evolution: Finding opponents worth beating. In S. Forrest, editor, *Proc. of the 6^{th} Int. Conf. Gen. Alg.*, pages 373–380, San Mateo,CA, 1995. Morgan Kaufman.
13. P. Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, 2000.
14. P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.