Team-Partitioned, Opaque-Transition Reinforcement Learning

Peter Stone and Manuela Veloso *
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213 {pstone,veloso}@cs.cmu.edu
http://www.cs.cmu.edu/{~pstone,~mmv}

In *RoboCup-98: Robot Soccer World Cup II*,
M. Asada and H. Kitano (eds.), 1999. Springer Verlag, Berlin.

Abstract

In this paper, we present a novel multi-agent learning paradigm called team-partitioned, opaque-transition reinforcement learning (TPOT-RL). TPOT-RL introduces the concept of using action-dependent features to generalize the state space. In our work, we use a learned action-dependent feature space. TPOT-RL is an effective technique to allow a team of agents to learn to cooperate towards the achievement of a specific goal. It is an adaptation of traditional RL methods that is applicable in complex, non-Markovian, multi-agent domains with large state spaces and limited training opportunities. Multi-agent scenarios are opaque-transition, as team members are not always in full communication with one another and adversaries may affect the environment. Hence, each learner cannot rely on having knowledge of future state transitions after acting in the world. TPOT-RL enables teams of agents to learn effective policies with very few training examples even in the face of a large state space with large amounts of hidden state. The main responsible features are: dividing the learning task among team members, using a very coarse, action-dependent feature space, and allowing agents to gather reinforcement directly from observation of the environment. TPOT-RL is fully implemented and has been tested in the robotic soccer domain, a complex, multi-agent framework. This paper presents the algorithmic details of TPOT-RL as well as empirical results demonstrating the effectiveness of the developed multi-agent learning approach with learned features.

1 Introduction

Reinforcement learning (RL) is an effective paradigm for training an artificial agent to act in its environment in pursuit of a goal. RL techniques rely on the premise that an agent's action policy affects its overall reward over time. As surveyed in [Kaelbling, Littman, & Moore, 1996], several popular RL techniques use dynamic programming to enable a single agent to learn an effective control policy as it traverses a stationary (Markovian) environment.

Dynamic programming requires that agents have or learn at least an approximate model of the state transitions resulting from its actions. Q-values encode future rewards attainable from neighboring states. A single agent can keep track of state transitions as its actions move it from state to state.

This paper focusses on teams of agents learning to collaborate towards a common goal in adversarial environments. While agents can still affect their reward through their actions, they can no longer necessarily track the team's state transitions: teammates and opponents affect and experience state transitions that are completely opaque to the agent. For example, an information agent may broadcast a message without any knowledge of who receives and reacts to it. In such *opaque-transition* settings, Q-values cannot relate to neighboring states (which are unknown), but must still reflect real-world long-term reward resulting from chosen actions.

While opaque-transition settings eliminate the possibility of using dynamic programming, they permit parallel learning among teammates. Since agents do not track state transitions, they can each explore a separate partition of the state space without any knowledge of state values in other partitions. This *team-partitioned* characteristic speeds

*This research is sponsored in part by the DARPA/RL Knowledge Based Planning and Scheduling Initiative under grant number F30602-95-1-0018. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the U. S. Government.

up learning by reducing the learning task of each agent. Nevertheless, the challenge of learning in a non-stationary (non-Markovian) environment remains.

This general setup builds upon the robotic soccer framework, which has been the substrate of our work. In more detail in this setup, agents' actions are *chained*, i.e., a single agent's set of actions allows the agent to select which other agent will be chained after in the pursuit to achieve a goal. A single agent cannot control directly the full achievement of a goal, but a chain of agents will. In robotic soccer the chaining of actions corresponds to passing a ball between the different agents. There are a variety of other such examples, such as information agents that may communicate through message passing. (These domains are for example in contrast with grid world domains in which a single agent moves from some initial location to some final goal location, domains where agents take actions in parallel though also possibly in coordination - two robots executing tasks in parallel, and game domains where the rules of the game enforce an agent and its opponent to alternate actions.) Because of our chaining of agents and the corresponding lack of control of single agents to fully achieve goals, we call these domains team-partitioned.

In addition, we assume that agents do not know the state that the world will be in after an action is selected, as another agent will *continue* the path to the goal. Adversarial agents can also intercept the chain and take control of the game. The domain becomes therefore opaque-transition. In short, We identify a way to do RL when the learning cannot even observe what state the team enters next, but the agent can use a reward function that captures a medium- to long-term result of the whole ensemble's learning.

In this paper we present team-partitioned, opaque-transition reinforcement learning (TPOT-RL). TPOT-RL can learn a set of effective policies with very few training examples. It relies on action-dependent dynamic features which coarsely generalize the state space. While feature selection is often a crucial issue in learning systems, our work uses a previously *learned* action-dependent feature. We empirically demonstrate the effectiveness of TPOT-RL in a multi-agent, adversarial environment, and show that the previously learned action-dependent feature can improve the performance of TPOT-RL.

The remainder of the paper is organized as follows. Section 2 formally presents the TPOT-RL algorithm. Section 3 details an implementation of TPOT-RL in the simulated robotic soccer domain with extensive empirical results presented in Section 4. Section 5 relates TPOT-RL to previous work and Section 6 concludes.

2 Team-Partitioned, Opaque-Transition RL

Formally, a policy is a mapping from a state space S to an action space A such that the agent using that policy executes action a whenever in state s. At the coarsest level, when in state s, an agent compares the expected, long-term rewards for taking each action $a \in A$, choosing an action based on these expected rewards. These expected rewards are learned through experience.

Designed to work in real-world domains with far too many states to handle individually, TPOT-RL constructs a smaller feature space V using action-dependent feature functions. The expected reward $Q(v, a)$ is then computed based on the state's corresponding entry in feature space.

In short, the policy's mapping from S to A in TPOT-RL can be thought of as a 3-step process:

State generalization: the state s is generalized to a feature vector v using the state generalization function $f : S \mapsto V$.

Value function learning: the feature vector v is used to estimate the expected reward for taking each possible action using the changing (learned) value function $Q : (V, A) \mapsto \mathbb{R}$.

Action selection: an action a is chosen for execution and its real-world reward is used to further update Q.

While these steps are common in other RL paradigms, each step has unique characteristics in TPOT-RL.

2.1 State Generalization

TPOT-RL's state generalization function $f : S \mapsto V$ relies on a unique approach to constructing V. Rather than discretizing the various dimensions of S, it uses *action-dependent* features. In particular, each possible action a_i is evaluated locally based on the current state of the world using a fixed function $e : (S, A) \mapsto U$. Unlike Q, e does not produce the expected long-term reward of taking an action; rather, it classifies the likely short-term effects of the action. For example, if actions sometimes succeed and sometimes fail to achieve their intended effects, e could indicate something of the following form: if selected, action a_7 is (or is not) likely to produce its intended effects.

In the multi-agent scenario, other than one output of e for each action, the feature space V also involves one coarse component that partitions the state space S among the agents. If the size of the team is m, then the partition function is $P : S \mapsto M$ with $|M| = m$. In particular, if the set of possible actions $A = \{a_0, a_1, \ldots, a_{n-1}\}$, then

$$\begin{aligned} f(s) &= \langle e(s,a_0), e(s,a_1), \ldots, e(s,a_{n-1}), P(s) \rangle, \text{ and so} \\ V &= U^{|A|} \times M. \end{aligned}$$

Thus, $|V| = |U|^{|A|} * m$. Since TPOT-RL has no control over $|A|$ or m, and since the goal of constructing V is to have a small feature space over which to learn, TPOT-RL will be more effective for small sets U.

This state generalization process reduces the complexity of the learning task by constructing a small feature space V which partitions S into m regions. Each agent need learn how to act only within its own partition. Nevertheless, for large sets A, the feature space can still be too large for learning, especially with limited training examples. Our particular action-dependent formulation allows us to reduce the effective size of the feature space in the value-function-learning step. Choosing features for state generalization is generally a hard problem. While TPOT-RL does not not specify the function e, our work uses a previously-learned dynamic feature function.

2.2 Value Function Learning

As we have seen, TPOT-RL uses action-dependent features. Therefore, we can assume that the expected long-term reward for taking action a_i depends only on the feature value related to action a_i. That is,

$$Q(\langle e(s,a_1), \ldots, e(s,a_{n-1}), P(s) \rangle, a_i) = Q(\langle e(s',a_1), \ldots', e(s',a_{n-1}), P(s') \rangle, a_i)$$

whenever $e(s,a_i) = e(s',a_i)$ and $P(s) = P(s')$. In other words, if $f(s) = v$, $Q(v, a_i)$ depends entirely upon $e(s,a_i)$ and is independent of $e(s,a_j)$ for all $j \neq i$.

Without this assumption, since there are $|A|$ actions possible for each feature vector, the value function Q has $|V| * |A| = |U|^{|A|} * |A| * m$ independent values. Under this assumption, however, the Q-table has at most $|A| * |U| * m$ entries: for each action possible from each position, there is only one relevant feature value. Therefore, even with only a small number of training examples available, we can treat the value function Q as a lookup-table without the need for any complex function approximation. To be precise, Q stores one value for every possible combination of action a, $e(s,a)$, and $P(s)$.

For example, Table 1 shows the entire feature space for one agent's partition of the state space when $|U| = 3$ and $|A| = 2$. There are $|U|^{|A|} = 3^2$ different entries in feature space with 2 Q-values for each entry: one for each possible action. $|U|^{|A|} * m$ is much smaller than the original state space for any realistic problem, but it can grow large quickly, particularly as $|A|$ increases. However, notice in Table 1 that, under the assumption described above, there are only $3 * 2$ independent Q-values to learn, reducing the number of free variables in the learning problem by 67% in this case.

$e(s,a_0)$	$e(s,a_1)$	$Q(v,a_0)$	$Q(v,a_1)$
u_0	u_0	$q_{0,0}$	$q_{1,0}$
u_0	u_1	$q_{0,0}$	$q_{1,1}$
u_0	u_2	$q_{0,0}$	$q_{1,2}$
u_1	u_0	$q_{0,1}$	$q_{1,0}$
u_1	u_1	$q_{0,1}$	$q_{1,1}$
u_1	u_2	$q_{0,1}$	$q_{1,2}$
u_2	u_0	$q_{0,2}$	$q_{1,0}$
u_2	u_1	$q_{0,2}$	$q_{1,1}$
u_2	u_2	$q_{0,2}$	$q_{1,2}$

$\Longrightarrow$

$e(s,a_i)$	$Q(v,a_0)$	$Q(v,a_1)$
u_0	$q_{0,0}$	$q_{1,0}$
u_1	$q_{0,1}$	$q_{1,1}$
u_2	$q_{0,2}$	$q_{1,2}$

Table 1: A sample Q-table for a single agent when $|U| = 3$ and $|A| = 2$: $U = \{u_0, u_1, u_2\}$, $A = \{a_0, a_1\}$. $q_{i,j}$ is the estimated value of taking action a_i when $e(s,a_i) = u_j$. Since this table is for a single agent, $P(s)$ remains constant.

The Q-values learned depend on the agent's past experiences in the domain. In particular, after taking an action a while in state s with $f(s) = v$, an agent receives reward r and uses it to update $Q(v,a)$ as follows:

$$Q(v,a) = Q(v,a) + \alpha(r - Q(v,a)) \tag{1}$$

Since the agent is not able to access its teammates' internal states, future team transitions are completely opaque from the agent's perspective. Thus it cannot use dynamic programming to update its Q-table. Instead, the reward r comes directly from the observable environmental characteristics—those that are captured in S—over a maximum number of time steps t_{lim} after the action is taken. The reward function $R : S^{t_{lim}} \mapsto \mathbb{R}$ returns a value at some time no further than t_{lim} in the future. During that time, other teammates or opponents can act in the environment and affect the action's outcome, but the agent may not be able to observe these actions. For practical purposes, it is crucial that the reward function is only a function of the observable world *from the acting agent's perspective*. In practice, the range of R is $[-Q_{max}, Q_{max}]$ where Q_{max} is the reward for immediate goal achievement .

The reward function, including t_{lim} and Q_{max}, is domain-dependent. One possible type of reward function is based entirely upon reaching the ultimate goal. In this case, an agent charts the actual (long-term) results of its policy in the environment. However, it is often the case that goal achievement is very infrequent. In order to increase the feedback from actions taken, it is useful to use an internal reinforcement function, which provides feedback based on intermediate states towards the goal. We use this internal reinforcement approach in our work.

2.3 Action Selection

Informative action-dependent features can be used to reduce the free variables in the learning task still further at the action-selection stage if the features themselves discriminate situations in which actions should not be used. For example, if whenever $e(s, a_i) = u_1$, a_i is not likely to achieve its expected reward, then the agent can decide to ignore actions with $e(s, a_i) = u_1$.

Formally, consider $W \subseteq U$ and $B(s) \subseteq A$ with $B(s) = \{a \in A | e(s, a) \in W\}$. When in state s, the agent then chooses an action from $B(s)$, either randomly when exploring or according to maximum Q-value when exploiting. Any exploration strategy, such as Boltzman exploration, can be used over the possible actions in $B(s)$. In effect, W acts in TPOT-RL as an action filter which reduces the number of options under consideration at any given time. Of course, exploration at the filter level can be achieved by dynamically adjusting W.

$e(s, a_0)$	$e(s, a_1)$	$Q(v, a_0)$	$Q(v, a_1)$
u_0	u_0	$q_{0,0}$	$q_{1,0}$
u_0	u_1	$q_{0,0}$	—
u_0	u_2	$q_{0,0}$	$q_{1,2}$
u_1	u_0	—	$q_{1,0}$
u_1	u_1	—	—
u_1	u_2	—	$q_{1,2}$
u_2	u_0	$q_{0,2}$	$q_{1,0}$
u_2	u_1	$q_{0,2}$	—
u_2	u_2	$q_{0,2}$	$q_{1,2}$

(a)

$e(s, a_0)$	$e(s, a_1)$	$Q(v, a_0)$	$Q(v, a_1)$
u_0	u_0	—	—
u_0	u_1	—	—
u_0	u_2	—	$q_{1,2}$
u_1	u_0	—	—
u_1	u_1	—	—
u_1	u_2	—	$q_{1,2}$
u_2	u_0	$q_{0,2}$	—
u_2	u_1	$q_{0,2}$	—
u_2	u_2	$q_{0,2}$	$q_{1,2}$

(b)

Table 2: The resulting Q-tables when **(a)** $W = \{u_0, u_2\}$, and **(b)** $W = \{u_2\}$.

For example, Table 2, illustrates the effect of varying $|W|$. In the rare event that $B(s) = \emptyset$, i.e. $\forall a_i \in A, e(s, a_i) \notin W$, either a random action can be chosen, or rough Q-value estimates can be stored using sparse training data. This condition becomes rarer as $|A|$ increases. For example, with $|U| = 3, |W| = 1, |A| = 2$ as in Table 2(b), 4/9 = 44.4% of feature vectors have no action that passes the W filter. However, with $|A| = 8$ only 256/6561 = 3.9% of feature vectors have no action that passes the W filter. If $|W| = 2$ and $|A| = 8$, only 1 of 6561 feature vectors fails to pass the filter. Thus using W to filter action selection can reduce the number of free variables in the learning problem without significantly reducing the coverage of the learned Q-table.

By using action-dependent features to create a coarse feature space, and with the help of a reward function based entirely on individual observation of the environment, TPOT-RL enables team learning in a multi-agent, adversarial environment even when agents cannot track state transitions.

3 TPOT-RL Applied to a Complex Multi-Agent Learning Task

Our research has been focussed on multi-agent learning in complex, collaborative and adversarial environments. Our general approach, called *layered learning*, is based on the premise that realistic domains are too complex for learning mappings directly from sensor inputs to actuator outputs. Instead, intermediate domain-dependent skills should be learned in a bottom-up hierarchical fashion [Stone & Veloso, 1998a]. We implemented TPOT-RL as the current highest layer of a layered learning system in the RoboCup soccer server [Noda, Matsubara, & Hiraki, 1996].

The soccer server used at RoboCup-97 [Kitano *et al.*, 1997] is a much more complex domain than has previously been used for studying multi-agent policy learning. With 11 players on each team controlled by separate processes; noisy, low-level, real-time sensors and actions; limited communication; and a fine-grained world state model including hidden state, the RoboCup soccer server provides a framework in which machine learning can improve performance. Newly developed multi-agent learning techniques could well apply in real-world domains.

A key feature of the layered learning approach is that learned skills at lower levels are used to train higher-level skills. For example, we used a neural network to help players learn how to intercept a moving ball. Then, with all players using the learned interception behavior, a decision tree (DT) enabled players to estimate the likelihood that a pass to a given field location would succeed. Based on almost 200 continuous-valued attributes describing teammate and opponent positions on the field, players learned to classify the pass as a likely success (ball reaches its destination or a teammate gets it) or likely failure (opponent intercepts the ball). Using the C4.5 DT algorithm [Quinlan, 1993], the classifications were learned with associated confidence factors. The learned behaviors proved effective both in controlled testing scenarios [Stone & Veloso, 1998a, Stone & Veloso, 1998c] and against other previously-unseen opponents in an international tournament setting [Kitano *et al.*, 1997].

These two previously-learned behaviors were both trained off-line in limited, controlled training situations. They could be trained in such a manner due to the fact that they only involved a few players: ball interception only depends on the ball's and the agent's motions; passing only involves the passer, the receiver, and the agents in the immediate vicinity. On the other hand, deciding where to pass the ball during the course of a game requires training in game-situations since the value of a particular action can only be judged in terms of how well it works when playing with particular teammates against particular opponents. For example, passing backwards to a defender could be the right thing to do if the defender has a good action policy, but the wrong thing to do if the defender is likely to lose the ball to an opponent.

Although the DT accurately predicts whether a player can execute a pass, it gives no indication of the strategic value of doing so. But the DT reduces a detailed state description to a single continuous output. It can then be used to drastically reduce the complex state and provide great generalization. In this work we use the DT as the crucial action-dependent feature function e in TPOT-RL.

3.1 State Generalization Using a Learned Feature

In the soccer example, we applied TPOT-RL to enable each teammate to simultaneously learn a high-level action policy. The policy is a function that determines what an agent should do *when it has possession of the ball*.[1] The input of the policy is the agent's perception of the current world state; the output is a target destination for the ball in terms of a location on the field, e.g. the opponent's goal. In our experiment, each agent has 8 possible actions as illustrated in Figure 1(a). Since a player may not be able to tell the results of other players' actions, or even when they can act, the domain is opaque-transition.

A team formation is divided into 11 positions ($m = 11$), as also shown in Figure 1(a) [Stone & Veloso, 1998c]. Thus, the partition function $P(s)$ returns the player's position. Using our layered learning approach, we use the previously trained DT as e. Each possible pass is classified as either a likely success or a likely failure with a confidence factor. Outputs of the DT could be clustered based on the confidence factors. In our experiments, we cluster into only two sets indicating success and failure. Therefore $|U| = 2$ and $V = U^8 \times \{PlayerPositions\}$ so $|V| = |U|^{|A|} * m = 2^8 * 11$. Even though each agent only gets about 10 training examples per 10-minute game and the reward function shifts as teammate policies improve, the learning task becomes feasible. This feature space is immensely smaller than the original state space, which has more than 22^{10^9} states.[2] Since e indicates the likely success or failure of each possible action, at action-selection time, we only consider the actions that are likely to succeed ($|W|$=1). Therefore, each player learns 8 Q-values, with a total of 88 learned by the team as a whole. Even with sparse training and shifting concepts, such a learning task is tractable.

---

[1] In the soccer server there is no actual perception of having "possession" of the ball. Therefore we consider the agent to have possession when it is within kicking distance.

[2] Each of the 22 players can be in any of 680*1050*3600 (x, y, θ) locations, not to mention the player velocities and the ball position and velocity.

3.2 Internal Reinforcement through Observation

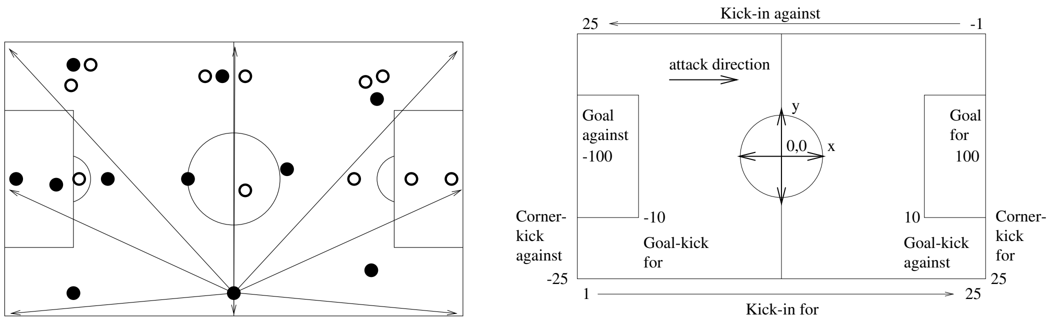
As in any RL approach, the reward function plays a large role in determining what policy is learned. One possible reward function is based entirely upon reaching the ultimate goal. Although goals scored are the true rewards in this domain, such events are very sparse. In order to increase the feedback from actions taken, it is useful to use an internal reinforcement function, which provides feedback based on intermediate states towards the goal. Without exploring the space of possible such functions, we created one reward function R.

R gives rewards for goals scored. However, players also receive rewards if the ball goes out of bounds, or else after a fixed period of time t_{lim} based on the ball's average lateral position on the field. In particular, when a player takes action a_i in state s such that $e(s, a_i) = u$, the player records the time t at which the action was taken as well as the x coordinate of the ball's position at time t, x_t. The reward function R takes as input the observed ball position over time t_{lim} (a subset of $S^{t_{lim}}$) and outputs a reward r. Since the ball position over time depends also on other agents' actions, the reward is stochastic and non-stationary. Under the following conditions, the player fixes the reward r:

1. if the ball goes out of bounds (including a goal) at time $t + t_o$ ($t_o < t_{lim}$);
2. if the ball returns to the player at time $t + t_r$ ($t_r < t_{lim}$);
3. if the ball is still in bounds at time $t + t_{lim}$.

In case 1, the reward r is based on the value r_o as indicated in Figure 1(b): $r = \frac{r_o}{1+(\phi-1)*t_o/t_{lim}}$. Thus, the farther in the future the ball goes out of bounds (i.e. the larger t_o), the smaller the absolute value of r. This scaling by time is akin to the discount factor used in Q-learning. We use $t_{lim} = 30 sec.$ and $\phi = 10$.

In cases 2 and 3, the reward r is based on the average x-position of the ball over the time t to the time $t+t_r$ or $t+t_{lim}$. Over that entire time span, the player samples the x-coordinate of the ball at fixed, periodic intervals and computes the average x_{avg} over the times at which the ball position is known. Then if $x_{avg} > x_t$, $r = \phi * \frac{x_{avg}-x_t}{x_{og}-x_t}$ where x_{og} is the x-coordinate of the opponent goal (the right goal in Figure 1(b)). Otherwise, if $x_{avg} \leq x_t$, $r = -\phi * \frac{x_t - x_{avg}}{x_t - x_{lg}}$ where x_{lg} is the x-coordinate of the learner's goal.[3] Thus, the reward is the fraction of the available field by which the ball was advanced, on average, over the time-period in question. Note that a backwards pass can lead to positive reward if the ball then moves forward in the near future.



(a) (b)

Figure 1: **(a)** The black and white dots represent the players attacking the right and left goals respectively. Arrows indicate a single player's action options when in possession of the ball. The player kicks the ball towards a fixed set of markers around the field, including the corner flags and the goals. **(b)** The component r_o of the reward function R based on the circumstances under which the ball went out of bounds. For kick-ins, the reward varies linearly with the x position of the ball.

The reward r is based on direct environmental feedback. It is a domain-dependent internal reinforcement function based upon heuristic knowledge of progress towards the goal. Notice that it relies solely upon the player's own impression of the environment. If it fails to notice the ball's position for a period of time, the internal reward is affected. However, players can track the ball much more easily than they can deduce the internal states of other players as they would have to do were they to determine future team state transitions.

As teammates learn concurrently, the concept to be learned by each individual agent changes over time. We address this problem by gradually increasing exploitation as opposed to exploration in all teammates and by using a learning rate $\alpha = .02$ (see Equation 1). Thus, even though we are averaging several reward values for taking an action in a

[3] The parameter ϕ insures that intermediate rewards cannot override rewards for attaining the ultimate goal.

given state, each new example accounts for 2% of the updated Q-value: rewards gained while teammates were acting more randomly are weighted less heavily.

4 Results

Empirical testing has demonstrated that TPOT-RL can effectively learn multi-agent control policies with very few training instances in a complex, dynamic domain. Figure 2(a) plots cumulative goals scored by a learning soccer team playing against an otherwise equally-skilled team that passes to random destinations over the course of a single long run equivalent in time to 160 10-minute games. In this experiment, and in all the remaining ones, the learning agents start out acting randomly and with empty Q-tables. Over the course of the games, the probability of acting randomly as opposed to taking the action with the maximum Q-value decreases linearly over periods of 40 games from 1 to .5 in game 40, to .1 in game 80, to point .01 in game 120 and thereafter. As apparent from the graph, the team using TPOT-RL learns to vastly outperform the randomly passing team. During this experiment, $|U| = 1$, thus rendering the function e irrelevant: the only relevant state feature is the player's position on the field.

A key characteristic of TPOT-RL is the ability to learn with minimal training examples. During the run graphed in Figure 2(a), the 11 players got an average of 1490 action-reinforcement pairs over 160 games. Thus, players only get reinforcement an average of 9.3 times each game, or less than once every minute. Since each player has 8 actions from which to choose, each is only tried an average of 186.3 times over 160 games, or just over once every game. Under these training circumstances, very efficient learning is clearly needed.

TPOT-RL is effective not only against random teams, but also against goal-directed, hand-coded teams. For testing purposes, we constructed an opponent team which plays with all of its players on the same side of the field, leaving the other side open as illustrated by the white team in Figure 1. The agents use a hand-coded policy which directs them to pass the ball up the side of the field to the forwards who then shoot on goal. The team periodically switches from one side of the field to the other. We call this team the "switching team."

Were the opponent team to always stay on the same side of the field, the learning team could always advance the ball up the other side of the field without any regard for current player positions. Thus, TPOT-RL could be run with $|U| = 1$, which renders e inconsequential. Indeed, we verified empirically that TPOT-RL is able to learn an effective policy against such an opponent using $|U| = 1$.

Against the switching team, a player's best action depends on the current state. Thus a feature that discriminates among possible actions dynamically can help TPOT-RL. Figure 2(b) compares TPOT-RL with different functions e and different sets W when learning against the switching team.

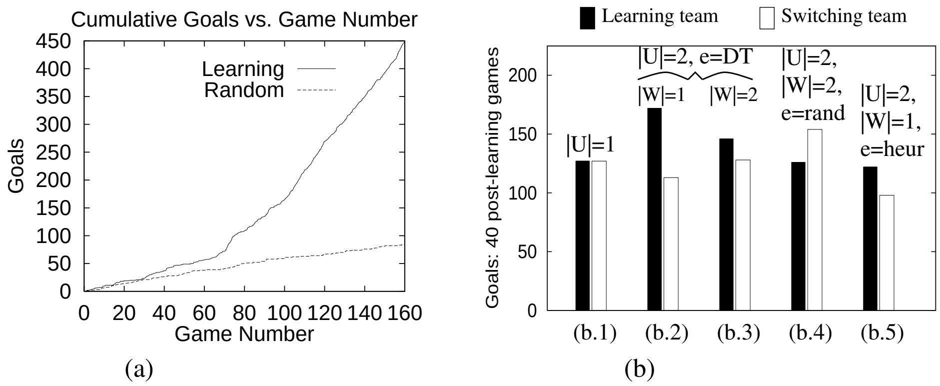


Figure 2: **(a)** Cumulative goals scored by a learning team playing against a randomly passing team. The independent variable is the number of 10-minute game intervals that have elapsed. **(b)** The results after training of 5 different TPOT-RL runs against the switching team.

With $|U| = 1$ (Figure 2(b.1)), the learning team is unable to capture different opponent states since each player has only one Q-value associated with each possible action, losing 139-127 (cumulative score over 40 games after 160 games of training). Recall that if $|U| = 1$ the function e cannot discriminate between different classes of states: we end up with a poor state generalization.

In contrast, with the previously trained DT classifying passes as likely successes or failures (e = DT) and TPOT-RL filtering out the failures, the learning team wins 172-113 (Figure 2(b.2)). Therefore the learned pass-evaluation feature is able to usefully distinguish among possible actions and help TPOT-RL to learn a successful action policy. The DT also helps learning when $W = U$ (Figure 2(b.3)), but when $|W| = 1$ performance is better.

Figure 2(b.4) demonstrates the value of using an informative action-dependent feature function e. When a random function e = rand is used, TPOT-RL performs noticeably worse than when using the DT. For the random e we show $|W| = 2$ because it only makes sense to filter out actions when e contains useful information. Indeed, when e = rand and $|W| = 1$, the learning team performs even worse than when $|W| = 2$ (it loses 167-60). The DT even helps TPOT-RL more than a hand-coded heuristic pass-evaluation function (e = heur) based on one that we successfully used on our real robot team [Veloso *et al.*, 1998] (Figure 2(b.5)).

Final score is the ultimate performance measure. However, we examined learning more closely in the best case experiment (e = DT, $|W| = 1$ — Figure 2(b.2)). Recall that the learned feature provides no information about which actions are *strategically* good. TPOT-RL must learn that on its own. To test that it is indeed learning to advance the ball towards the opponent's goal (other than by final score), we calculated the number of times each action was predicted to succeed by e and the number of times it was actually selected by TPOT-RL after training. Throughout the entire team, the 3 of 8 actions towards the opponent's goal were selected 6437/9967 = 64.6% of the times that they were available after filtering. Thus TPOT-RL learns that it is, in general, better to advance the ball towards the opponent's goal.

To test that the filter was eliminating action choices based on likelihood of failure we found that 39.6% of action options were filtered out when e = DT and $|W| = 1$. Out of 10,400 actions, it was never the case that all 8 actions were filtered out.

5 Discussion and Related Work

TPOT-RL brings together several techniques that have been proposed theoretically or tested in simple domains. This section highlights the components of TPOT-RL and relates them to previous work.

Typical RL paradigms update the value of a state-action pair based upon the value of the subsequent state (or state distribution). As presented in [Kaelbling, Littman, & Moore, 1996], the typical update function in Q-learning is $Q(s, a) = Q(s, a) + \alpha(r + \gamma max_a Q(s', a) - Q(s, a))$ where s' is the state next reached after executing action a in state s and γ is the discount factor. While characterized as "model-free" in the sense that the agent need not know the transition function $T : (S, A) \mapsto S$, these paradigms assume that the agent can observe the subsequent state that it enters.

However, the vast amount of hidden state coupled with the multi-agent nature of this domain make such a paradigm impossible for the following reasons. Having only local world-views, agents cannot reliably discern when a teammate is able to take an action. Furthermore, even when able to notice that a teammate is within kicking distance of the ball, the agent certainly cannot tell the feature values for the teammate's possible actions. Worse than being model-free, multi-agent RL must deal with the inability to even track the team's state trajectory. Thus we use Equation 1, which doesn't rely on knowing s'.

Notice that the opaque-transition characteristic also does not fit into the partially observable Markov decision process (POMDP) framework [Kaelbling, Cassandra, & Littman, 1994]. While POMDPs deal with hidden state, they do assume that the agent at least knows when it has transitioned to a new state and may act again.

The construction of feature space V can have a huge effect on the nature of Q. For example, in [Salustowicz, Wiering, & Schmidhuber, 1998], a grid-like discretization is used for V. Since too many states result for a lookup-table, a neural network is used as the value function approximator. This approach is shown not to work very well, and the authors conclude that a more complex function approximator might work better. In contrast, we take the approach of using a smaller feature space and the simplest possible evaluation function: a lookup-table. We can do so by relying on our layered learning approach to learn the state generalization function.

The use of machine learning in multi-agent systems has recently been receiving a good deal of attention. For an extensive survey, see [Stone & Veloso, 1997]. This section highlights the work most related to TPOT-RL.

The internal reinforcement in the reward function R is similar to Mataric's progress estimators [Mataric, 1994]. There, the short-term real-world effects of actions are used as an intermediate reward to help robots reach the ultimate goal location. Mataric's *conditions* also play a similar role to the features used here, reducing the size of the evaluation function domain. This work was in a completely collaborative, as opposed to our adversarial, setting.

Previous multi-agent reinforcement learning systems have typically dealt with much simpler tasks than the one presented here. Littman uses Markov games to learn stochastic policies in a very abstract version of 1-on-1 robotic soccer [Littman, 1994]. There have also been a number of studies of multi-agent reinforcement learning in the pursuit domain [Arai, Miyazaki, & Kobayashi, 1997, Tan, 1993]. In this domain, four predators chase a single prey in a small grid-like world.

Also in a predator-like task, [Zhao & Schmidhuber, 1996] uses a single run to deal with the opponents' shifting policies and ignore the opponents' policies just as we do. The effects of opponent actions are captured in the reward function.

Another team-partitioned, opaque transition domain is network routing as considered in [Boyan & Littman, 1994]. Each network node is considered as a separate agent which cannot see a packet's route beyond its own action. A major difference between that work and our own is that neighboring nodes send back their own value estimates whereas we assume that agents do not even know their neighboring states. Thus unlike TPOT-RL agents, the nodes are able to use dynamic programming.

In other soccer systems, there have been a number of learning techniques that have been explored. However, most have learned low-level, individual skills as opposed to team-based policies [Asada *et al.*, 1996, Stone & Veloso, 1998b]. Interestingly, [Luke *et al.*, 1998] uses genetic programming to evolve team behaviors from scratch as opposed to our layered learning approach.

6 Conclusion

TPOT-RL is an adaptation of RL to non-Markovian multi-agent domains with opaque transitions, large state spaces, hidden state and limited training opportunities. The fully implemented algorithm has been successfully tested in simulated robotic soccer, such a complex multi-agent domain with opaque transitions. TPOT-RL facilitates learning by partitioning the learning task among teammates, using coarse, action-dependent features, and gathering rewards directly from environmental observations. Our work uses a learned feature within TPOT-RL.

TPOT-RL represents the third and currently highest layer within our ongoing research effort to construct a complete learning team using the layered learning paradigm. As advocated by layered learning, it uses the previous learned layer—an action-dependent feature—to improve learning. TPOT-RL can learn against any opponent since the learned values capture opponent characteristics. The next learned layer could learn to choose among learned team policies based on characteristics of the current opponent. TPOT-RL represents a crucial step towards completely learned collaborative and adversarial strategic reasoning within a team of agents.

Acknowledgements

We thank Justin Boyan, Andrew Moore, and Astro Teller for helpful comments and suggestions.

References

[Arai, Miyazaki, & Kobayashi, 1997] Arai, S.; Miyazaki, K.; and Kobayashi, S. 1997. Generating cooperative behavior by multi-agent reinforcement learning. In *Sixth European Workshop on Learning Robots*.

[Asada *et al.*, 1996] Asada, M.; Noda, S.; Tawaratumida, S.; and Hosoda, K. 1996. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning* 23:279–303.

[Boyan & Littman, 1994] Boyan, J. A., and Littman, M. L. 1994. Packet routing in dynamically changing networks: A reinforcement learning approach. In Cowan, J. D.; Tesauro, G.; and Alspector, J., eds., *Advances In Neural Information Processing Systems 6*. Morgan Kaufmann Publishers.

[Kaelbling, Cassandra, & Littman, 1994] Kaelbling, L. P.; Cassandra, A. R.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*.

[Kaelbling, Littman, & Moore, 1996] Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285.

[Kitano *et al.*, 1997] Kitano, H.; Kuniyoshi, Y.; Noda, I.; Asada, M.; Matsubara, H.; and Osawa, E. 1997. RoboCup: A challenge problem for AI. *AI Magazine* 18(1):73–85.

[Littman, 1994] Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, 157–163. San Mateo, CA: Morgan Kaufman.

[Luke *et al.*, 1998] Luke, S.; Hohn, C.; Farris, J.; Jackson, G.; and Hendler, J. 1998. Co-evolving soccer softbot team coordination with genetic programming. In Kitano, H., ed., *RoboCup-97: Robot Soccer World Cup I*, 398–411. Berlin: Springer Verlag.

[Mataric, 1994] Mataric, M. J. 1994. Interaction and intelligent behavior. MIT EECS PhD Thesis AITR–1495, MIT AI Lab.

[Noda, Matsubara, & Hiraki, 1996] Noda, I.; Matsubara, H.; and Hiraki, K. 1996. Learning cooperative behavior in multi-agent environment: a case study of choice of play-plans in soccer. In *PRICAI'96: Topics in Artificial Intelligence (Proc. of 4th Pacific Rim International Conference on Artificial Intelligence, Cairns, Australia)*, 570–579.

[Quinlan, 1993] Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

[Salustowicz, Wiering, & Schmidhuber, 1998] Salustowicz, R. P.; Wiering, M. A.; and Schmidhuber, J. 1998. Learning team strategies: Soccer case studies. *Machine Learning*. To appear.

[Stone & Veloso, 1997] Stone, P., and Veloso, M. 1997. Multiagent systems: A survey from a machine learning perspective. Technical Report CMU-CS-97-193, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

[Stone & Veloso, 1998a] Stone, P., and Veloso, M. 1998a. A layered approach to learning client behaviors in the RoboCup soccer server. *Applied Artificial Intelligence* 12:165–188.

[Stone & Veloso, 1998b] Stone, P., and Veloso, M. 1998b. Towards collaborative and adversarial learning: A case study in robotic soccer. *International Journal of Human-Computer Studies* 48(1):83–104.

[Stone & Veloso, 1998c] Stone, P., and Veloso, M. 1998c. Using decision tree confidence factors for multiagent control. In Kitano, H., ed., *RoboCup-97: Robot Soccer World Cup I*. Berlin: Springer Verlag. 99–111.

[Tan, 1993] Tan, M. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, 330–337.

[Veloso *et al.*, 1998] Veloso, M.; Stone, P.; Han, K.; and Achim, S. 1998. The CMUnited-97 small-robot team. In Kitano, H., ed., *RoboCup-97: Robot Soccer World Cup I*. Berlin: Springer Verlag. 242–256.

[Zhao & Schmidhuber, 1996] Zhao, J., and Schmidhuber, J. 1996. Incremental self-improvement for life-time multi-agent reinforcement learning. In *Proceedings of the 4th International Conference of Simulation of Adaptive Behaviros (SAB 4)*, 363–372. MIT Press.