Plays as Effective Multiagent Plans Enabling Opponent-Adaptive Play Selection

Michael Bowling
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
bowling@cs.ualberta.ca

Brett Browning and **Manuela Veloso**
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3891 USA
{brettb,mmv}@cs.cmu.edu

Abstract

Coordinated action for a team of robots is a challenging problem, especially in dynamic, unpredictable environments. Robot soccer is an instance of a domain where well defined goals need to be achieved by multiple executors in an adversarial setting. Such domains offer challenging multiagent planning problems that need to coordinate multiagent execution in response to other agents that are not part of our team plans. In this work, we introduce the concept of a *play* as a multiagent plan that combines both reactive principles, which are the focus of traditional approaches for coordinating robot actions, and deliberative principles. We further introduce the concept of a *playbook* as a method for seamlessly combining multiple team plans. The playbook provides a set of alternative team behaviors which form the basis for our third contribution of *play adaptation*. We describe how these concepts were concretely implemented in the CMDragons robot soccer team. We also show empirical results indicating the importance of adaptation in adversarial or other unpredictable environments.

Introduction

Coordination and adaptation are two of the most critical challenges for deploying teams of robots to perform useful tasks, in domains where there are clear goals to be achieved. There are many instances of single robot systems that can successfully achieve goals using reactive action selection methods. When controlling *teams* of robots, it is far more difficult to design reactive action selection approaches that coordinate the execution of multiple robots. It is even more challenging when the team actions need to be selected to respond to great uncertainty in the domain. One possible source of this uncertainty is if the domain involves other agents, particularly adversarial ones, that are not under the team's control. In this paper, we examine the challenge of controlling a team of robots within the context of robot soccer, a multi-robot, goal-driven, adversarial domain. The presence of adversarial robots creates significant uncertainty for predicting the outcome of actions. This is particularly true if the opponents' behavior and capabilities are unknown, as is the case in a robot soccer competition. This lack of knowledge about the opponent makes it impossible to predefine a model of the domain that can be used for planning as in classical planning. As such, robot soccer contains many of the challenges described above, as well as issues found in other realistic multi-robot settings.

Despite the inherent unpredictability of the adversarial domain, most robot soccer approaches involve single, static, monolithic team strategies (e.g., see robot team descriptions in (Birk, Coradeschi, & Tadokoro 2002).) Although these strategies entail complex combinations of reactive and deliberative approaches, they can still perform poorly against unknown opponents or in unexpected situations. With the uncertainty present in robot soccer, such situations are common, and there have been examples of seemingly superior teams being defeated by unexpected opponent play.

An alternative approach uses models of opponent behavior, constructed either before or during the competition (Intille & Bobick 1999), which are used then to determine the best team response. A model may be used in a reactive fashion to trigger a pre-coded static strategy, or in a deliberative fashion through the use of a planner (Riley & Veloso 2002). Although these techniques have had success, they have limitations such as the requirement for an adequate representation of opponent behavior. For a completely unknown opponent team, constructing an accurate model of their strategy is not likely to be practical.

In this work we take a novel approach to adapting to the opponent that is based on observing our own team's effectiveness rather than observing the opponent's behavior. We replace a single monolithic team strategy, with multiple team plans that are appropriate for different opponents and situations, which we call *plays*. Each play represents a predefined deliberative multiagent plan as a coordinated sequence of team actions, and is explicit enough to facilitate evaluation of that play's execution. A *playbook* encapsulates the plays that are available to the team, and provides multiple strategic options. Each execution of a play from the playbook can then be evaluated and this information collected for aid in future play selection. Successful plays, whose successes may be attributed to weaknesses in the opponent or particular strengths of our team, are selected more often, while unsuccessful plays are ignored.

We begin by providing an overview of our CMDragons'03 robot soccer team and the role of plays in the team's

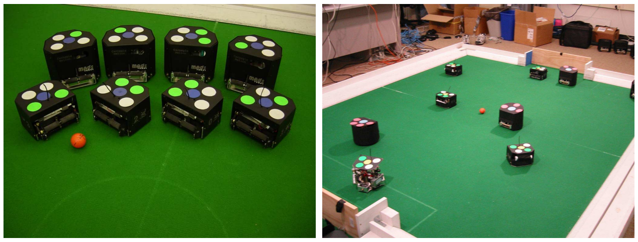

Figure 1: The picture on the left shows our small-size robots, and the picture on the right shows the team in action.

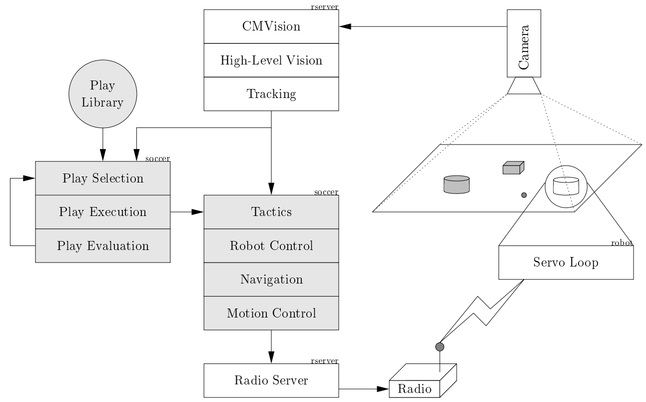

Figure 2: Overview of the CMDragons'02 team architecture.

decision making. We then present plays and the play execution system in detail, followed by a description of how the team adaptively selects plays. We then evaluate the effectiveness of the play-based strategy and adaptive play selection by examining its effectiveness in recent competitions. Finally, we conclude.

Overview

In this section, we briefly describe our CMDragons'03 robot soccer team, the basis for the work explored in this paper. This overview focuses on how the strategy system, described in the next section, interacts with the system as a whole.

The CMDragons are a team of Small-Size League (SSL) soccer robots that participated at RoboCup 2003. Figure 1 shows pictures of robots that comprise the team. SSL robot soccer, part of the RoboCup initiative (Kitano *et al.* 1997), consists of two teams of five robots that play soccer with an orange golf-ball on a 2.8m by 2.3m field surrounded by short, sloped walls. Rules are based on human soccer and are enforced by a human referee. Robots must conform to size and shape specifications, but no standard platforms exist. SSL is characterized by the allowance of cameras mounted above the field for shared global perception and additional off-field computation resources making a team as a whole autonomous, rather than individual robots. SSL robots are typically fast, cruising at speeds of 1–2m/s while the ball moves at over 4m/s, and occasionally much faster. This makes SSL an environment that requires fast response, good long-term strategy, strong individual robot skills and capable multi-robot coordination, in order for a team to be successful.

Figure 2 shows the major components of the CMDragons' system. Stacked boxes define a component of the system where the boxes define a flow of control from top to bottom. The arrows show the flow of information. The control loop, synchronized with image frames at 30Hz, consists of taking an image from the camera via the framegrabber and processing it, determining the control for each robot on the team, and sending these velocity commands using radio communication to the robots. Each robot operates local servo loops to carry out the commands. Due to space limitations, we refer the reader to our earlier works (Bruce *et al.* 2002) and (Bruce & Veloso 2002), for a more thorough description of the entire system. Here we focus on the tactics and strategy layers of the control software, the shaded regions in Figure 2.

The tactics and strategy layers make up the bulk of decision-making in the system. The tactics layer, the stack of boxes on the right, encompasses individual robot skills. For each frame, each robot executes a single tactic independently of the others. The strategy layer, the stack of boxes on the left, provides the coordination mechanism and executes one instance for the entire team. Thus, the strategy layer must meld individual robot skills into powerful and adaptable team behavior.

Tactics are defined to be any behavior executable by a single robot. Table 1 shows the list of implemented tactics. Each tactic is highly parameterized and performs a complex, single robot task that itself may consist of many sub-policies run in succession. Each tactic makes use of a robot control layer that maintains robot-specific information persistent even after a robot's tactic changes. The layer transforms tactic commands into target points for navigation. The navigation layer then produces short term, obstacle-free waypoints for the motion control system using a fast randomized planner. These waypoints are used by the motion control module to generate the actual velocity commands sent to the robot.

A tactic, therefore, is a complex interaction between low-level navigation and motion control, and higher-level skill-based code. For example, consider the `position_for_deflection` tactic. The tactic itself determines the best location within a region for deflecting passes into the goal. This requires sampling of points over the specified region and evaluation of the deflection angles using a deflection heuristic. The best evaluated point is then fed into the navigation layer, and in turn to the motion control layer, to generate the actual commands necessary to drive the robot to the calculated position.

Play-Based Strategy

The main question addressed in this work is: "Given a set of effective and parameterized individual robot behaviors, how do we select each robot's behavior (possibly using past execution experience) to achieve the team's goals?" This is the problem addressed by our strategy component, which is di-

Active Tactics:

```
shoot (Aim | Noaim | Deflect ⟨role⟩)
steal [⟨coordinate⟩]
clear
active_def [⟨coordinate⟩]
pass ⟨role⟩
dribble_to_shoot ⟨region⟩
dribble_to_region ⟨region⟩
spin_to_region ⟨region⟩
receive_pass
receive_deflection
dribble_to_position ⟨coordinate⟩ ⟨theta⟩
position_for_kick
```

Non-active Tactics:

```
position_for_loose_ball ⟨region⟩
position_for_rebound ⟨region⟩
position_for_pass ⟨region⟩
position_for_deflection ⟨region⟩
defend_line ⟨p1⟩ ⟨p2⟩ ⟨min-dist⟩ ⟨max-dist⟩
defend_point ⟨p1⟩ ⟨min-dist⟩ ⟨max-dist⟩
defend_lane ⟨p1⟩ ⟨p2⟩
block ⟨min-dist⟩ ⟨max-dist⟩ ⟨side-pref⟩
mark ⟨orole⟩ (ball | our_goal | | shot)
goalie
stop
velocity ⟨vx⟩ ⟨vy⟩ ⟨vtheta⟩
position ⟨coordinate⟩ ⟨theta⟩
```

Table 1: List of tactics along with their parameters.

agrammed by the left-most shaded components of Figure 2. Our team strategy utilizes the concept of a *play* as a team plan with multiple plays collected into a *playbook*.

Goals

The main evaluation criterion for team strategy is performance, i.e., goal achievement. However, a single, static, monolithic team strategy that maximizes performance is impractical. Indeed, in an adversarial domain with an unknown opponent, a single static optimal strategy is unlikely to exist. Therefore we have broken down the performance criterion into easier to achieve subgoals. The goals of a strategy system are:

1. Coordinate team behavior,
2. Execute temporally extended sequences of action,
3. Allow for special behavior for certain circumstances,
4. Allow ease of human design and augmentation,
5. Enable exploitation of short-lived opportunities, and
6. Allow on-line adaptation to the specific opponent.

The first four goals require plays to be able to express complex, coordinated, and sequenced behavior among teammates. In addition, plays must be human readable to make strategy design and modification simple. These goals also require a powerful system capable of executing the complex behaviors the play describes. The fifth goal requires the execution system to also recognize and exploit opportunities that are not explicitly described by the current play. Finally, the sixth goal requires the system to alter its behavior over time. Notice that these goals, although critical to the robot soccer task, are also of general importance for the coordination of agent teams in other adversarial or unpredictable environments. We have developed a play-based team strategy, using a specialized play language, to meet these goals. In the following sections, we will describe the three major components of the play-based strategy engine: play specification using the play language, the play execution system, and the playbook adaptation mechanism used to autonomously alter team strategy to a specific opponent during a competition.

Table 2: A simple example of a play.

```
PLAY Naive Offense

APPLICABLE offense
DONE aborted !offense

ROLE 1
  shoot A
  none
ROLE 2
  defend_point {-1400 250} 0 700
  none
ROLE 3
  defend_lane {B 0 -200} {B 1175 -200}
  none
ROLE 4
  defend_point {-1400 -250} 0 1400
  none
```

Play Specification

Plays are specified using a play language in an easy-to-read text format (e.g., Tables 2 and 3). A simple example of a play is shown in Table 2. Plays use keywords, denoted by all capital letters, to mark different pieces of information. Each play has two components: *basic information* and *role information*. The basic information describes when a play can be executed ("`APPLICABLE`"), when execution of the play should stop ("`DONE`"), and some execution details (e.g., "`FIXEDROLES`", "`TIMEOUT`", and "`OROLE`"). The role information ("`ROLE`") describes how the play is executed, making use of the tactics from Table 1. We describe these keywords below.

Applicability. The `APPLICABLE` keyword denotes when a play can be executed. What follows the keyword is a conjunction of high-level predicates that all must be true for the play to be considered executable. Multiple `APPLICABLE` keywords can be used to denote different disjunctive conditions for when the play may be executed. This allows plays to effectively specify when they can be executed as a logical

DNF of high-level predicates. In the example play in Table 3, the play can only be executed when the `offense` predicate is true. The `offense` predicate is a complex function of the present and past history of possession of the ball and the ball's field position. New predicates are easily added to the system, but many useful predicates related to possession, ball position, and the other team's robots already exist.

A play's applicability condition is very similar to operator preconditions in classical planning. By constraining the applicability of a play, we can design special purpose plays for very specific circumstances. Table 3 shows an example of a complex play that uses the `in_their_corner` predicate to constrain the play to execute only when the ball is in one of the opponent team's corners. The play explicitly involves dribbling the ball out of the corner to get a better angle for a shot on the goal.

Termination. Unlike classical planning, the level of uncertainty in this task makes it difficult to predict the outcome of a particular plan. Therefore a play does not have effects, but rather has something similar called termination conditions. Termination conditions are specified by the keyword `DONE` followed by a result (e.g., `aborted`) and a conjunctive list of high-level predicates similar to the applicability conditions. Plays may have multiple `DONE` conditions, each with a different result, and a different conjunction of predicates. Whenever one of these `DONE` conditions are satisfied, the play is terminated, and a new play must be selected. In the example play in Table 3, the only terminating condition is if the team is no longer on offense. In this case the play's result is considered to have been `aborted`.

The results for plays are: `succeeded`, `completed`, `aborted`, and `failed`. These results are used to evaluate the success of the play for the purposes of reselecting the play later. This is the major input to the play adaptation system which we describe in the next section.

There are three other ways in which plays can be terminated. The first is due to referee signals that change the game state. The second is when the sequence of behaviors defined by the play are `completed`. These two are described in more detail with the play execution system below. The third occurs when a play runs for too long without terminating. A timeout causes the play to terminate with an `aborted` result and a new play is selected. Thus, the team commits to a course of action, but if no progress is made due to unforeseen circumstances, another approach will be tried. The timeout period has a team configurable default value, however, a play may use the `TIMEOUT` keyword to override this default timeout limit (e.g., Table 3).

Roles. Roles are the action component of a play, and each play has four roles corresponding to the non-goalie robots on the field. Each role contains a list of tactics (also called behaviors) with associated parameters for the robot to perform in sequence. As tactics are heavily parameterized, the range of tactics can be combined into nearly an infinite number of play possibilities. Table 3 shows an example play where the first role executes two sequenced tactics. First the robot dribbles the ball out of the corner and then switches to the shooting behavior. Meanwhile the other roles execute a single behavior for the play's duration.

```
PLAY Two Attackers, Pass from Corner

APPLICABLE offense in_their_corner
DONE aborted !offense

TIMEOUT 15

OROLE 0 closest_to_ball

ROLE 1
  pass 3
  mark 0 from_shot
  none
ROLE 2
  block 320 900 -1
  none
ROLE 3
  position_for_pass { R { B 1000 0 } ...
  receive_pass
  shoot A
  none
ROLE 4
  defend_line { -1400 1150 } ...
  none
```

Table 3: A complex play involving restricted applicability conditions and sequencing of behaviors.

Sequencing implies an enforced synchronization, or coordination between roles. Once a tactic completes, all roles move to their next behavior in their sequence (if one is defined). For example, consider the complex play shown in Table 3. In this play the player assigned to pass the ball completes the pass, then it switches to the mark behavior. The receiver of the pass will simultaneously switch to receive the pass, after which it will try to execute the shooting tactic.

Opponent Roles. Some behaviors are dependent on the positions of specific opponents on the field. Opponent roles are used to identify a specific opponent based on an evaluation method for the tactic to use. The example in Table 3, shows an opponent role defined using the `OROLE` keyword and the `closest_to_ball` method. Thus, the first role will try to mark the opponent closest to the ball away from the ensuing shot, after executing the pass.

Coordinate Systems. Parameters for tactics are also very general by allowing for a variety of coordinate systems in specifying points and regions on the field. Coordinates may be specified as absolute field positions or ball relative field positions. In addition, a coordinate system's positive y-axis can be oriented to point toward the side of the field that the ball is on, the side of field the majority of the opponents are on, or even a careful combination of these two factors. This allows tremendous flexibility in the specification of the behaviors used in plays and prevents unnecessary duplication of plays for symmetric field situations.

Play Execution

The play execution module is responsible for instantiating the active play into actual robot behavior. Instantiation consists of many key decisions: role assignment, role switching, sequencing tactics, opportunistic behavior, and termination.

Role assignment is dynamic, rather than being fixed, and is determined by tactic-specific methods. To prevent conflicts, assignment is prioritized by the order in which roles appear. Thus, the first role, which usually involves ball manipulation, is assigned first and considers all four field robots. The next role is assigned to one of the remaining robots, and so on. This prioritization provides the execution system the knowledge to select the best robots to perform each role and also provides the basis for role switching. Role switching is a very effective technique for exploiting changes in the environment that alter the effectiveness of robots fulfilling roles. The executor continuously reexamines the role assignment for possible opportunities to improve it as the environment changes. Although, it has a strong bias toward maintaining the current assignment to avoid oscillation.

Sequencing is needed to move the entire team through the list of tactics in sequence. When the tactic executed by the *active player*, the robot whose role specifies a tactic related to the ball (see Table 1), succeeds then the play transitions *each* role to the next tactic in their sequence.

Opportunistic behavior accounts for unexpected situations where a very basic action would have a valuable outcome. For example, the play executor evaluates the duration of time and potential success of each robot shooting immediately. If a robot can shoot quickly enough and with a high likelihood of success, it will immediately switch its behavior to take advantage of the situation. Thus, opportunistic behavior enables plays to have behavior beyond that specified explicitly. As a result, a play can encode a long sequence of complex behavior without encumbering its ability to respond to unexpected short-lived opportunities.

Finally, the play executor checks the play's termination criteria, the completion status of the tactics, and the incoming information from the referee. If the final active tactic in the play's sequence of tactics completes then the play terminates as `completed`. If the game is stopped by the referee for a goal, penalty, or free kick, the play terminates. The outcome of the play depends upon the condition. Fouls and penalty kicks result in a success or failure as appropriate. A free kick results in a completion or an abort as appropriate.

Play Selection

The final facet of the playbook strategy system is play selection and the related problem of adapting play selection during the course of a game. Our basic selection scheme uses the applicability conditions for each play to form a candidate list from which one play is selected at random. To adapt play selection, we modify the probability of selecting an applicable play based on past experience of play execution. In the next section, we describe this scheme. This is followed by experimental results showing the usefulness of a playbook approach and the effectiveness of the adaptation.

Playbook Adaptation

The problem of selecting which play to execute and adapting the selection during the game based on experience, is similar to a traditional on-line learning task, called the "experts" problem, or k-armed bandits problems. We quickly review work on experts algorithms and then explain how the developed techniques can be applied to selecting plays.

In a standard experts task, a decision-maker must select among a set of experts' advice to follow. After selecting an expert, the payoff of all of the experts are revealed and the decision-maker appropriately rewarded. The decision-maker performs this selection repeatedly. The usual measure of performance in this task is that of *regret*. Let $r^t_{i=1\ldots n}$ be the reward at time t for expert i, and x^t be the expert selected by some algorithm at time t. Then the regret of the algorithm at time T is,

$$\textsc{Regret}^T = \max_{i=1\ldots n} \sum_{t=0}^{T} r_i^t - \sum_{t=0}^{T} r_{x^t}^t .$$

In words, regret is the amount of additional reward that could have been received if the algorithm had known which expert would be the best and chose that expert at each decision point. With only a minor assumption about the experts' payoffs over time, there exist algorithms where average regret in the limit goes to zero, i.e.,

$$\lim_{T\to\infty} \frac{\textsc{Regret}^T}{T} = 0.$$

One well-known algorithm with this property is Littlestone and Warmuth's randomized weighted majority (1994).

The experts task is nearly identical to the play selection problem. Each play is an "expert" recommending a course of action. After a play terminates, the team must select among its available plays, and after that play terminates, its outcome (e.g., goal, penalty kick, free kick, etc.) is the reward received for executing that play. The team must now select a new play, and this repeats throughout the game. The notion of regret also has applicability in the play context. A selection algorithm with zero-regret in the limit guarantees that the team over time is doing at least as well as its most effective play, which is not known in advance. This would give a very powerful guarantee on the team's ability to adapt to an initially unknown opponent.

There are two additional complications preventing the application of a standard no-regret experts algorithm. First, only the reward of the play selected is observed rather than the rewards of all of the plays that could have been selected. Second, not all plays are applicable at each decision point. The first complication has been addressed in an algorithm by Auer and colleagues (1995) called Exp3. Exp3 is a modification of their experts algorithm, Hedge, which has a very simple formulation. At time T, choose expert i with the following probability,

$$Pr(x^T = i) = \frac{e^{R_i^{T-1}}}{\sum_{j=1}^{n} e^{R_j^{T-1}}},$$

where R_i^T is the cumulative reward received by expert i up to decision point T,

$$R_i^T = \sum_{t=0}^{T} r_i^t.$$

Auer and colleagues proved that Hedge's regret goes to zero in the limit. Exp3 uses the following modification. Hedge is run with $\hat{r}_i^t$ in place of the true r_i^t, and defined as,

$$\hat{r}_i^t = \begin{cases} r_i^t / Pr(x^t = i) & \text{if } x^t = i \\ 0 & \text{otherwise.} \end{cases}$$

The basic idea is that observed rewards are scaled inversely with the probability that the observation was gained, i.e., the probability of selecting that expert. Essentially, this estimates the total reward the expert would have received had it been chosen at each decision point, as Hedge expects.

The second complication is that not all experts are available, since a play's applicability conditions may not be satisfied in the current state. Therefore, not only will no observation of the play's possible outcome be observed, but the play could not even be selected. This situation is known as sleeping experts, and is accompanied with a redefinition of regret. The following analysis comes from Freund and colleagues (1997). Let a_i^t be 1 if expert i is awake (i.e., applicable) at time t, and 0 otherwise. Also, let $\Delta(n)$ be the set of probability distributions over the n experts (i.e., the n-dimensional simplex). Define sleeping experts' regret as,

$$\text{SREGRET}^T = \left(\max_{x \in \Delta(n)} \sum_{t=1}^{T} \sum_{i=1}^{n} a_i^t \left(\frac{x(i)}{\sum_{j=1}^{n} x(j) a_j^t} \right) r_i^t \right) - \sum_{t=0}^{T} r_{x^t}^t.$$

In words, the new notion of regret is the amount of additional reward that could have been received if the best (in hindsight) distribution over experts was chosen at each decision point, where the probabilities are normalized to choose among those experts that are awake.

It is possible to have sleeping regret go to zero in the limit, and involves a second modification to a basic expert algorithm, such as Hedge. The modification requires the following invariant: if expert i is sleeping at time t, then,

$$Pr(x^{t+1} = i) = Pr(x^t = i),$$

that is, sleeping does not affect the expert's probability of being chosen in the future. From our perspective of using such a selection algorithm for plays, this is an ideal invariant since it means that special purpose plays which are only applicable in a few circumstances, are neither penalized nor rewarded for their selectivity.

We can modify Exp3 to make this invariant hold, by requiring the sum of the exponentials of the R_i^t's for the experts that are awake to remain constant after updating. Since it will be easier to focus on the exponentials, let $w_i^t \equiv e^{R_i^t}$, which we will call weights. Notice that adding r_i^t to R_i^t is the same as multiplying w_i^t by $e^{r_i^t}$. Let this factor be called the multiplier, or m_i^t. Exp3 modified to handle sleeping experts can now be defined. The choice at a particular decision point is defined as,

$$Pr(x^t = i) \quad = \quad \frac{a_i^t w_i^t}{\sum_j a_j^t w_j^t}. \tag{1}$$

The weights are then updated as follows,

$$\hat{w}_i^t \quad = \quad w_i^{t-1} (m_i^t)^{1/Pr(x^t = i)} \tag{2}$$

$$w_i^t \quad = \quad \hat{w}_i^t \cdot N_i^t, \tag{3}$$

where,

$$N_i^t = \begin{cases} 1 & \text{if } a_i^t = 0 \\ \frac{\sum_j a_j^t w_j^{t-1}}{\sum_j a_j^t \hat{w}_j^t} & \text{otherwise.} \end{cases}$$

If the expert is sleeping, w_i^t remains unchanged. If it is awake, then it is updated as with Exp3 (Equation 2), but all the awake experts' w_i^t's are normalized to have a constant sum (Equation 3).

It is the sleeping experts version of Exp3 that we used in our CMDragons'03 team. The main decision left in its application is where the multipliers m_i come from. They obviously depend on the outcome of the play, but multipliers still need to be assigned to particular outcomes. We simply chose these values with the principle that successes and failures should be symmetric, and that the multipliers should aggressively change behavior so they can have an impact over the short time span of a game. In competitions we used a multiplier of $3/2$ for successes and $2/3$ for failures, and $11/10$ for completions and $10/11$ for aborts.

An example of adaptation is shown in Table 4. In subtable (a) we have a list of three plays, each with an initial weight of 1, so that each play is equally likely to be selected. At a particular decision point, though, only two of these plays may be applicable as shown in the table. Subtable (b) shows the intermediate weight computation after `Offense 2` was selected and scored a goal (multiplier of 1.5), i.e., the result of Equation 2. Subtable (c) shows the final weights after normalization, i.e., the result of Equation 3. There are two things to notice: (1) the play's success increases its probability of being selected in the future, (2) the non-applicability of `Corner` does not affect its probability of being selected when applicable (i.e., the probability remains 0.33 if all plays are applicable both before and after the observed success).

Evaluation

The play strategy architecture described here has been used for team control for the CMDragons team at *(i)* RoboCup 2002 held in Fukuoka, Japan (reached quarterfinals,), *(ii)* RoboCup American Open'03 held in Pittsburgh, USA (champions), *(iii)* and RoboCup 2003 held in Padua,

(a)

Play	w_i^0	App?	$Pr(x^0 = i)$
`Offense 1`	1	✓	0.5
`Offense 2`	1	✓	0.5
`Corner`	1		0

(b)

Play	$\hat{w}_i^1$	App?	$Pr(x^1 = i)$
`Offense 1`	1	✓	0.22
`Offense 2`	2.25	✓	0.56
`Corner`	1	✓	0.22

(c)

Play	w_i^1	App?	$Pr(x^1 = i)$
`Offense 1`	0.62	✓	0.21
`Offense 2`	1.38	✓	0.46
`Corner`	1	✓	0.33

Table 4: An example of adaptation execution. Lists the current, intermediate, and resulting weights associated with different plays. Also lists the probabilities associated with each play given whether the play is applicable ("App?").

Italy (reached semi-finals). This provides a body of concrete experience in real competitions to evaluate our approach.

We have previously evaluated plays as a means to coordinate a robot team in controlled simulation experiments (Bowling *et al.* 2003). This work demonstrated that plays indeed captured critical strategic choices. We executed four different offensive plays against three simple defensive behaviors and showed that the optimal play to select depended critically on the defensive behavior. We also demonstrated that a simple multiplicative weight update, used at RoboCup 2002, could be used to adapt play selection and improve team performance even in the short confines of a single game. We noticed in these experiments, though, that the multiplicative scheme could fall into certain traps, which motivated formulating adaptation as an experts problem. The more recent competitions, which used the sleeping experts based update, is the focus of evaluation here.

To complete our analysis of the performance of the playbook architecture, we focus on answering four questions:

1. Do plays allow for rapid generation of team behavior by a human "coach"?
2. Do plays allow for synchronized team actions in a realistic robot setting?
3. Does play adaptation work against real teams and within the time-limits of a real game?

To answer the first question, we rely on anecdotal accounts of our use of plays at RoboCup competitions. Looking back over the playbooks used at each RoboCup event, it is rare to find that the playbook remains static from game to game. In some rare cases, the *entire* playbook was rewritten prior to the game. Thus, the entire team strategy which typically consists of around 15 plays, was redesigned and tested within a few hours. In practice, most time goes into devising and testing new plays as opposed to the mechanics of writing the team strategy in the play language.

Plays explicitly allow for complex synchronized team plans. The question naturally arises whether in a stochastic team environment it is possible to execute such complex sequences with any reliability. Figure 3 shows the successful execution of a "one-shot" deflection play from the game against Toin Albatross at RoboCup 2003. The sequence requires one robot to move to the ball while its teammate positions itself so that it has a shot on goal and can receive a pass. The teammate with the ball must then kick the ball toward its teammate, hitting a 6cm target from a distance of about 1.5m. The receiving teammate must then time its kick on the ball to direct it toward the goal. Due to the high kicking speeds (4m/s), the entire sequence will take at most 2s to execute. Thus, we conclude that complex sequences can be executed even in such a stochastic environment.

We lastly focus on play adaptation. Let us analyze the four attacking plays used against the same team as before, Toin Albatross. The final score was 10-0 to our team CMDragons (games stop when the goal differential reaches 10 goals). Although there were 16 plays used in total, we only discuss the offensive plays to maintain the clarity of the discussion. The four plays are:

- **Play 1:** 1 Attacker, 2 Point Defenders, 1 Deep Defender
- **Play 2:** 1 Attacker (Deflections), 2 Point Defenders, 1 Deep Defender
- **Play 3:** 1 Attacker, 3 Deep Staggered Defenders
- **Play 4:** 2 Attackers (Deflections), 2 Deep Defenders

Plays 1 and 2 are identical except the second allows for "one-shot" deflections, while play 1 does not. Otherwise, these two plays have one attacker aggressively going for the ball. Two rear defenders stay far back near the goalie. The fourth player stays near the halfway line in an aggressive but defensive position. Thus, the plays are conservative and shield the goal from powerful long range kicks but still maintain the ability to counterattack. Play 3 uses an even more conservative defensive pattern where the fourth player moves deep closer to the goal keeper. The last play uses an aggressive attack where one player goes to the ball and the other comes forward into an offensive position ready for passes to shoot at the goal. The two defenders assume a defensive pattern like that for plays 1 and 2. Plays 2 and 4 both allow for deflections while plays 1 and 3 do not.

Figure 4 shows the weight values. Clearly, plays 2 and 3 never prove successful against the opponent. In contrast, play 4 initially proves successful leading to 3 goals and it quickly dominates the selection. However, toward the end of the half, play 1 suddenly scores a goal. Due to its low rate of selection prior to this point its weight is boosted by a large amount. It then proceeds to become the dominant play for the remainder of the game (not shown here). Whether play 1 suddenly proved successful due to a change in the opponent's game play (a timeout occurred in the neighborhood of the change) or simply unlucky results earlier is not clear. It is clear that this play is preferred to play 4, due to the riskiness of attempted deflections, which can miss and go out-of-bounds. Since play 1 scores without the additional risk it is the optimally performing play. This example demonstrates the ability of the play selection to identify successful plays and alter the team's behavior to exploit these discoveries even within the short confines of a single game.

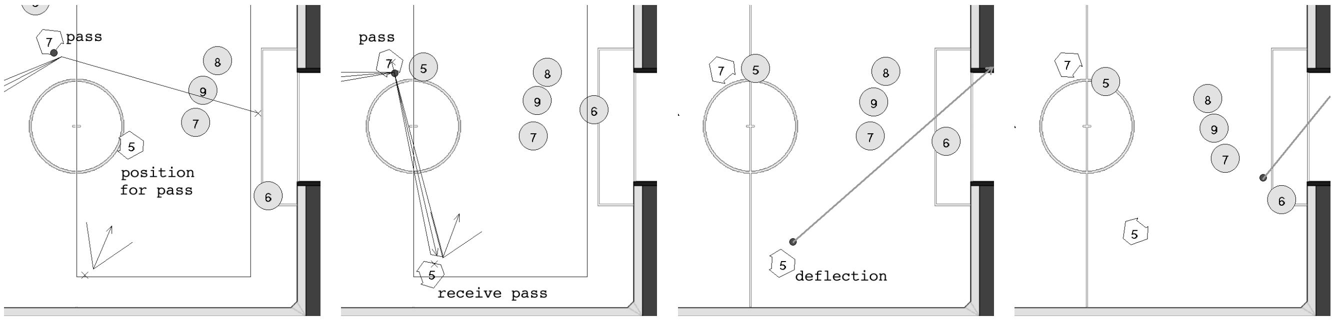

Figure 3: Example of a deflection goal against Toin Albatross. The dark lines show debugging output from the tactics and the light line shows the tracked ball velocity. The first image shows the shooting robot unable to take a shot, robot 5 begins moving to a good deflection point. The second image shows the kicker lined up and its target zone on robot 5. Image final two images show the kick and resulting deflection to score a goal. The entire sequence takes less than one second.

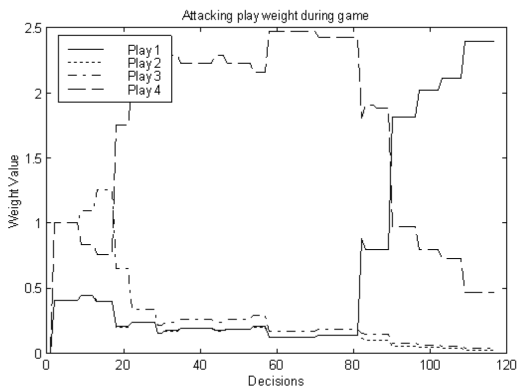

Figure 4: Weight changes for attacking plays during the match against Toin Albatross.

Conclusion

In conclusion, we have introduced a novel team strategy engine based on the concept of a play as a team plan, which can be easily defined by a play language. Multiple, distinct plays can be collected into a playbook where mechanisms for adapting play selection can enable the system to improve the team response to an opponent without prior knowledge of the opponent's strategy. We adapted an experts algorithm to the problem of sequential play selection and demonstrate its capability of repeatedly selecting effective plays during the course of a single game. The system was fully implemented for our CMDragons robot soccer system and tested at RoboCup 2002, 2003, and the American Open. Possible future directions of research include extending the presented play language, enhancing the play adaptation algorithm, and in general investigating further applications of reinforcement learning to the team coordination problem.

Acknowledgments

This research was supported by U.S. Air Force Grants Nos. F30602-98-2-0135 and F30602-00-2-0549 and by U.S. Army Grant No. DABT63-99-1-0013. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, by the U.S. Air Force, the U.S. Army, or the U.S. Government.

References

Auer, P.; Cesa-Bianchi, N.; Freund, Y.; and Schapire, R. E. 1995. Gambling in a rigged casino: The adversarial multi-arm bandit problem. In *36th Annual Symposium on Foundations of Computer Science*, 322–331. Milwaukee, WI: IEEE Computer Society Press.

Birk, A.; Coradeschi, S.; and Tadokoro, S., eds. 2002. *RoboCup 2001: Robot Soccer World Cup V*.

Bowling, M.; Browning, B.; Chang, A.; and Veloso, M. 2003. Plays as team plans for cooperation and adaptation. In *the IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modelling, Planning, Learning, and Communicating*.

Bruce, J., and Veloso, M. 2002. Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002*, 2383–2388.

Bruce, J.; Bowling, M.; Browning, B.; and Veloso, M. 2002. Multi-robot team response to a multi-robot opponent team. In *ICRA Workshop on Multi-Robot Systems*.

Freund, Y.; Schapire, R. E.; Singer, Y.; and Warmuth, M. K. 1997. Using and combining predictors that specialize. In *Proceedipngs of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, 334–343.

Intille, S., and Bobick, A. 1999. A framework for recognizing multi-agent action from visual evidence. In *AAAI-99*, 518–525. AAAI Press.

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.

Littlestone, N., and Warmuth, M. 1994. The weighted majority algorithm. *Information and Computation* 108:212–261.

Riley, P., and Veloso, M. 2002. Planning for distributed execution through use of probabilistic opponent models. In *ICAPS-02*.