Planning for Distributed Execution Through Use of Probabilistic Opponent Models

Patrick Riley and **Manuela Veloso**
Carnegie Mellon University
{pfr,mvv}@cs.cmu.edu
http://www.cs.cmu.edu/~{pfr,mmv}

Abstract

In multiagent domains with adversarial and cooperative team agents, team agents should be adaptive to the current environment and opponent. We introduce an online method to provide the agents with team plans that a "coach" agent generates in response to the specific opponents. The coach agent can observe the agents' behaviors but it has only periodic communication with the rest of the team. The coach uses a Simple Temporal Network to represent team plans as coordinated movements among the multiple agents and the coach searches for an opponent-dependent plan for its teammates. This plan is then communicated to the agents, who execute the plan in a distributed fashion, using information from the plan to maintain consistency among the team members. In order for these plans to be effective and adaptive, models of opponent movement are used in the planning. The coach is then able to quickly select between different models online by using a Bayesian style update on a probability distribution over the models. Planning then uses the model which is found to be the most likely. The system is fully implemented in a simulated robotic soccer environment. In several recent games with completely unknown adversarial teams, the approach demonstrated a visible adaptation to the different teams.

Introduction

Multiagent domains can include team and adversarial agents. One of the main challenges of such domains is the coordination and the response of teammates to the adversarial agents. This paper contributes our work to address this problem. We use the concrete simulated robotic soccer platform, which is a rich multiagent environment, including fully distributed agents in two different teams of up to eleven agents. Of additional interest is a "coach" agent for each team. The coach has a centralized view of the world, and it can communicate with its teammates, but only at occasional times. In the robotic soccer domain, these times are when the game is stopped for some reason. This type of scenario is one instance of the *Periodic Synchronization Domains* (Stone & Veloso 1999) where team agents can periodically synchronize their team strategies.

Our work is driven by the goal of significantly improving the performance of teams of agents through their adaptation and effective response to different adversarial teams. Indeed an excellent robotic soccer team should change strategies as a function of the adversary that it plays against. Extensive work has been done on adapting the teammates' behavior to their opponents, mainly at the individual low-level of positioning and interactions between a small number of agents (Veloso, Bowling, & Stone 1999; Stone, Riley, & Veloso 2000). The procedure for these positioning adaptations does not change throughout a game, limiting the degree to which the team adapts to the opponent team's behaviors.

This paper reports our work on going one additional major step towards the adaptive response of teammates to the opponent, by gathering and responding to the opponents' behaviors throughout the game. We specifically focus on responding effectively after the game is stopped, in what are known as setplay situations. Several preset setplay plans (Stone, Riley, & Veloso 2000; Stone, Veloso, & Riley 1999; Veloso *et al.* 1998) have been introduced which indeed provide great opportunities to position the teammates strategically and have been shown to impact the performance of a team. In this work, we contribute *adaptive* setplays which change and improve throughout a game as a function of and in response to the opponent team's behavior.

We use our coach agent that compiles the necessary overall view of how the opponent team behaves. The coach communicates to its teammates a team plan which is executed and monitored in a fully distributed manner. In a nutshell, here is the complete overview of our ATAC approach, standing for **A**daptive **T**eam-**A**dversarial **C**oaching, as reported in this paper. The coach agent is equipped with a number of pre-defined opponent models. These models are probabilistic representations of predicted opponents locations. The models can then be matched to observed movements of the opponent agents. The coach continues to gather observations and when the game is stopped, e.g., due to an out-of-bound call, the coach rapidly takes advantage of the short available time to create a team setplay plan that is a function of the matched modeled opponent's behavior. The plan is generated through a hill-climbing search in plan space using an evaluation function that embeds the predictions of the opponent model perceived to be the most likely during the game. The plan generation, in addition to the recognition of the opponents' model, notably uses the model *to predict* the opponent agents' behaviors. In addition to this plan creation algorithm per se, one additional challenge of this research was the selection and use of an appropriate plan representation. We use a Simple Temporal Network based representation that effectively captures the temporal dependencies between the plan steps, and, most importantly, explicitly records bounds on the expected execution times of the actions. The setplay plans, as generated and delivered by the coach to the teammates, notably include the necessary information for the agents to execute and monitor the team plan in a fully distributed manner. The coach also observes the execution of the plan generated in order to update the selection of an appropriate model for the current opponent. Our overall ATAC approach created a variety of setplay plans in adaptation to completely unknown opponent teams.

The Environment

ATAC is fully implemented in the Soccer Server System (Noda *et al.* 1998) as used in RoboCup (Kitano *et al.* 1997). The Soccer Server System is a server-client system which simulates soccer between distributed agents. Clients communicate using a standard network protocol with well-defined actions. The server keeps track of the current state of the world, executes the actions which the clients request, and periodically sends each agent noisy, incomplete information about the world. Agents receive noisy information about the direction and distance of objects on the field (the ball, players, goals, etc.); information is provided only for objects in the field of vision of the agent.

There are 11 independent players on each side, as well as a "coach" agent who has a global view of the world, but whose only action is to send short messages to the players while the ball is out of play.

Actions must be selected in real-time, with each of the agents having an opportunity to act 10 times a second. Each of these action opportunities is known as a "cycle." Visual information is sent 6 or 7 times per second. Over a standard 10 minute game, this gives 6000 action opportunities and 4000 receipts of visual information. In order to handle the real-time constraints, efficient abstraction of important information is needed.

The agents can communicate, but the communication is of limited bandwidth and unreliable. Further, only agents close to the player who is talking will be able to hear the message. The coach is able to communicate with *all* of the players on the field, when communication from the coach is available.

A few points which are especially related to this work should be noted. The score and global clock are the only shared state features. For all other information, the agents must rely on their own local view. Because of noise in localization and sensing of objects, there is inconsistent knowledge of the locations of objects. It is possible to make this information mostly consistent among the agents, but it takes time for the agents to communicate. Another crucial point of our special interest is the presence of an active and intelligent opponent team.

Technical details about the Soccer Server System can be found at (Rob 2001).

Multi-Agent Plans

Plan Representation

A pictorial representation of the type of plan generated is shown in Figure 1. A teammate starts with the ball at the bottom of the figure. It passes the ball up and to the right, and simultaneously the middle teammate moves to receive the pass. Also, the other teammate at the top of the figure simultaneously moves to position itself to receive the next pass. When the middle teammate starts that pass, the top agent moves to receive it.

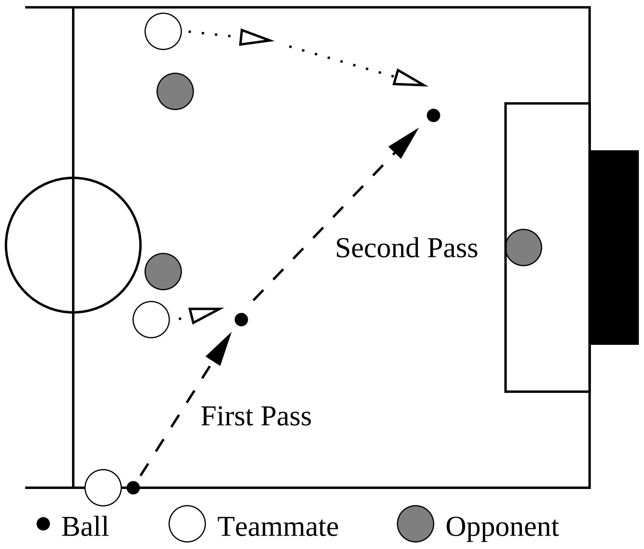

Figure 1: An Example Plan

A plan is represented as a Simple Temporal Network, as introduced in (Dechter, Meiri, & Pearl 1991). A Simple Temporal Network (or STN for short) is a directed graph that represents the temporal constraints between events. Each node in the graph represents an event, and each edge represents a temporal constraint between the events.

For example, in Figure 2, there are "Start Pass", "Start Goto", "End Pass", "End Goto", and "Initial Position" nodes, representing the beginning and ending of events. Each node also has information associated with it (not shown) about which agent is responsible for bringing about this event and details about the event. For example, a "Start Goto" node is labeled with the location where the agent should go. Note also that the "Initial Position" node is a privileged node which must be executed by all of the agents first. The edges' labels are of the form $[a, b]$. For an edge from event e_1 to e_2, this means that e_2 must occur *at least* a time steps after e_1, but *no more than* b time steps. Note that these values can be negative (which means that e_2 can occur *before* e_1), as well as infinity.

The nodes represent events which the agents must bring about through their actions. Events in general have the following properties:

Type Specifies a domain-specific type of event. The types of events we use are described below. The type determines several important properties for the plan execution,

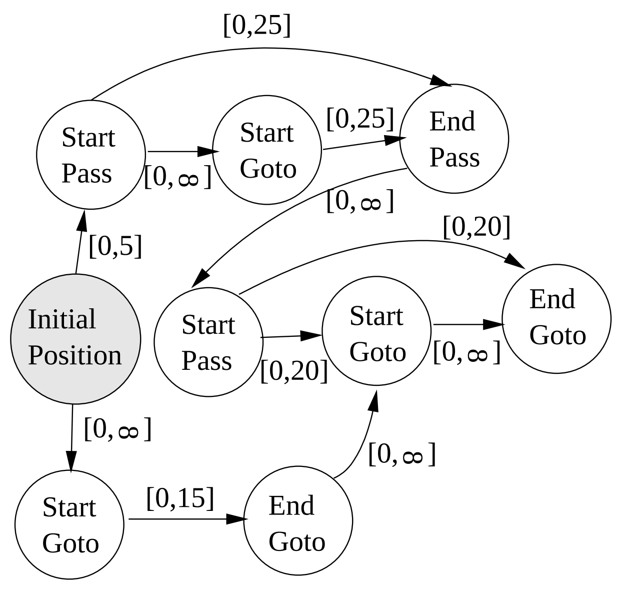

Figure 2: An Example Simple Temporal Network (STN)

including precondition monitors and execution test functions.

Agents Indicates which agent(s) are responsible for bringing about the occurrence of this event.

Node Pointer Some nodes require pointers to other nodes in the plan. These pointers are not edges in the graph. They are a way for a single copy of information (such as an agent list) to exist in the network and have that information easily accessible by several nodes.

Additional Data Other information may be required for particular applications. In this work, many events also have locations associated with them. Details on how these locations are used are indicated below.

In the robotic soccer domain, we use the following events:

Initial Position$(a_1, \ell_1, a_2, \ell_2, \dots)$ Describes which agents $(a_1, a_2, \dots)$ are involved in the set play and the locations on the field where they should start $(\ell_1, \ell_2, \dots)$. This is also the root node for execution; this event must occur before any other. This node is completed when the play begins.

Start Goto(a, ℓ) Specifies the location ℓ to which agent a should start moving. This node is completed once the agent has begun to move to ℓ. This node is also used to indicate the receiver of a pass.

End Goto(p) Represents the conclusion of the move began by the "Start Goto" node pointed to by p. This node is completed when the agent gets to the location ℓ (from the "Start Goto" node).

Start Pass(a, ℓ) Represents an agent a kicking the ball to a location ℓ (where another agent should receive it). It is completed once the ball leaves the agent's virtual foot.

End Pass(p_1, p_2) Represents the conclusion of a pass. In particular, p_1 points to a "Start Pass" node (for the kicker) and p_2 points to a "Start Goto" node (for the receiver). This event is completed when the receiver controls the ball.

Clear Ball(a, ℓ) Represents an agent a kicking the ball to a location ℓ. This differs from a "Start Pass" node because no particular agent is expected to get the ball. There is no associated end node, because the plan is always complete after a "Clear Ball" node executes.

Plan execution basically consists of the agents performing the events (being careful to respect the temporal constraints, see below), and observing the world to know the current state of the plan. Once the agent is no longer responsible for bringing about the occurrence of any events in the plan (i.e. it is not in the agent list for any uncompleted node), it considers its role complete.

The authors are not aware of any other work where STNs are used as a multi-agent plan representation. However, there are several features of STNs which make them appealing for multi-agent plans.

- The network very naturally represents the parallelism of agents' actions. The temporal constraints express some basic needed coordination between the agents.
- Temporal constraints can be used to help agents detect failures in the plan. For example, in the top three nodes of the plan in Figure 2, if the agent intended to receive the pass has not received the ball by 25 cycles after the ball was kicked, then the plan is declared failed. The agent can then abort the plan and use other reactive behaviors.
- If an event e is not ready to execute because of temporal constraints, the network represents which event(s) are preventing e from being ready. This means that if an agent is responsible for executing an event that is not ready, it can determine where in the world to look to see the event holding e (the event which is preventing e from being ready). For example, in some cases a "Start Goto" node is constrained to occur after a "Start Pass" node. The agent for the "Start Goto" node knows that it must look at the agent which is starting the pass in order to determine when to start moving.

Plan Execution

Muscettola, Morris, & Tsamardinos have described a "dispatching execution" algorithm for STN execution (Muscettola, Morris, & Tsamardinos 1998). This allows easy and efficient propagation of temporal constraints through the network at the cost of adding edges. The first step is to construct the "all-pairs" closure of the STN (i.e. make an edge from every node a to every other node b whose length is the shortest path from a to b). Muscettola *et al* describe a method to then prune some of those edges to reduce the time requirements of plan execution, which is important for STNs consisting of thousands of nodes. However, since we are working with networks of tens of nodes instead of thousands, we do not prune any edges.

Our execution algorithm extends the dispatching execution algorithm to the multi-agent case. We use the dispatching algorithm (unchanged) to maintain information about time bounds for executions of events. The agents execute the following steps at every cycle:

1. **Global Monitors**: Each agent checks the world state for conditions which indicate plan failure, such as the opposing agents controlling the ball.
2. **Node Completion**: Each agent looks at every node to determine if the conditions for that event being completed have occurred. For example, an "End Pass" event has occurred if the receiving agent has control of the ball. That information is passed to the dispatching execution algorithm and a check is done for out of order execution or other violated temporal constraints.
3. **Find Node to Execute**: Each agent locates the node N which it is next required to execute. If there is no such node, the agent's role in the plan is complete. We require that the nodes for a particular agent be fully ordered, which means that at any time at most one node for an agent will be ready to execute.
4. **Node Precondition Monitor**: Some event nodes have constraint monitors (or preconditions) on their execution. For example, the "Start Pass" node requires that the agent believes the teammate intended to receive the ball will be able to get it. This decision is based on a learned decision tree (Stone 2000) or other analytic methods (McAllester & Stone 2000). If N has such constraints, they are verified here.
5. **Watch Constraining Node**: If temporal constraints (from the dispatching algorithm) indicate that N is not ready to execute because another node N' must execute first, then the agent looks to the location on the field where N' should execute.
6. **Wait for Time**: If temporal constraints (from the dispatching algorithm) indicate that N is not ready to execute simply because more time needs to pass, the agent scans the field to monitor the state of the world.
7. **Execute Node**: Otherwise, the agent works towards executing N. This may consist of kicking the ball, turning towards a target, running to a point, etc. All of these behaviors uses the reactive CMUnited99 (Stone, Riley, & Veloso 2000) layer to get robust performance of such commands as "get the ball" or "kick the ball hard in direction x."

Several features of the soccer simulation environment are different from the other environments in which STNs have typically been used. We chose to use STNs because of their rich temporal expressibility. Much previous work (such as (Intille & Bobick 1999)) has more limited temporal expressibility in plans. First, and most importantly is the multi-agent aspect. STNs have typically been used solely to help in the scheduling of interrelated tasks, not in the coordination between agents. A related problem caused by noisy, partial observability is that the agents may have different ideas about when an event actually occurred. Further, even the agent which is executing an event can not precisely control when the event will occur. For example, the motion of kicking in a particular direction is a complicated series of turns, dashes, and weak and powerful kicks. There is no obvious way to predict how long it will take the reactive CMUnited99 behavior to execute to completion. Lastly, all execution is noisy in space also. If an agent tries to kick in a particular direction, the ball will only approximately head that direction. This means the agents have to be flexible in plan execution to what actually occurs. If the temporal constraints are too tight, even mostly normal executions will be aborted.

In short, we use the STN plan to track two separate questions: "What should each agent be doing at each time step?" and "What have all the other agents done up to this time step?" In most STN applications, only the first question is addressed; the executor is assumed to be the only agent. Morris and Muscettola (Morris & Muscettola 2000) have considered allowing there to be some uncertainty in the temporal execution of some events. However, they are more concerned with guaranteeing that execution will be successful, while we are more interested in the optimistic possibility that there is still some valid execution in the given temporal constraints. Also, they do not address the multi-agent aspects which are important to this work.

Plan Creation

We divide the process of plan creation into four steps:

1. **Waypoint Planning**: This module plans the ball's trajectory as a series of straight line segments (i.e. passes). Constraints are put on the length of the segments, but agent movements are not part of this step.
2. **Role Making**: Given the planned trajectory for the ball, this step creates roles for agents.
3. **STN Compilation**: This step takes the output from the role making step and turns it into a Simple Temporal Network.
4. **Agent Instantiation**: At this point, actual agents are assigned to the roles in the plan.

Waypoint Planning

In order to plan the waypoints, we use models of opponent movement. The next section fully describes these models, but for the purposes of this section, we can just that say that waypoint planning is a path planning problem with straight-line segments, and dynamic, probabilistic obstacles (the obstacles are the opponents). The sense of "probabilistic" here is that the obstacles are not fixed regions, but rather we have a probability distribution over their locations.

Part of what makes this problem challenging is that it is not even clear what it means to have a solution. There may be one path that is extremely long, and goes through an obstacle with very low probability, and another path that is much shorter but has a higher probability of going through an obstacle. To decide which path is better requires a trade-off in the length and safety of the path. Further, we do not have a single goal position, but rather an ordered set of positions.

These constraints make it difficult to apply many of the traditional path planning methods, such as those described by Latombe(1991). Planning methods which deal with uncertainty do not usually handle obstacles whose location is only probabilistic. Rather, they are more focused on dealing with noisy execution when following the path. Approaches which deal with moving obstacles do not address uncertainty in obstacle location.

The D^* algorithm, developed by Stentz (1994) was also considered. However, D^* is mostly useful for replanning when obstacles are observed to move, not handling the up-front probabilistic movements we model here.

In order to plan in this challenging domain, we decided to directly write an evaluation function for paths and use hill-climbing on a set of paths to find a locally optimal path. We first describe the hillclimbing algorithm and then describe the evaluation function.

Our path planning algorithm is then as follows:

```
$S$ := Set of starting paths
while (there is time left)
    Pick a random path $p \in S$
    Pick a random point $x$ on $p$
    Randomly set $A$ to true or false
    $\forall$ small displacement vector $v$
        Make path $p'$ by moving $x$ by $v$
        If (A)
            In $p'$ move pts.  after $x$ by $v$
        If $p'$ has higher value, set $p = p'$
```

Note that sometimes (decided by the variable A) we move only a single point in a hillclimbing step and sometimes we move the entire tails of paths. We found experimentally that using both techniques allows us to escape more local minima.

The hillclimbing runs for a fixed number of cycles. Halfway through that time the set S is reduced to just the path with the current best evaluation. This allows the second half of the hillclimbing to focus on improving a single path as much as possible.

The set of starting paths are preset and depend on the location of the ball. This allows us to very effectively incorporate domain knowledge into the planner. An example of such knowledge is that it is better to get the ball downfield and for the ball not to be in front of our own goal. We effectively give a set of possible “shapes” for the path and the waypoint planner optimizes among and around those choices.

The crucial part for the hillclimbing method is the evaluation function. We use the following weighted factors:

- Ball control at end

  If the last segment of the path is a pass, we are in control of the ball at the end of the play (this has value 1.0). If the last segment of the pass is a send (kicking the ball downfield with no specific agent designated to get the ball), this has value 0.0.

- Ball’s end location

  The value here depend on whether we are in control of the ball at the end of the play. In general getting the ball near the goal and the opponent’s baseline has high value, and just getting the ball further downfield is also of high value. A sample of the function is shown in Figure 3.

- Length of path

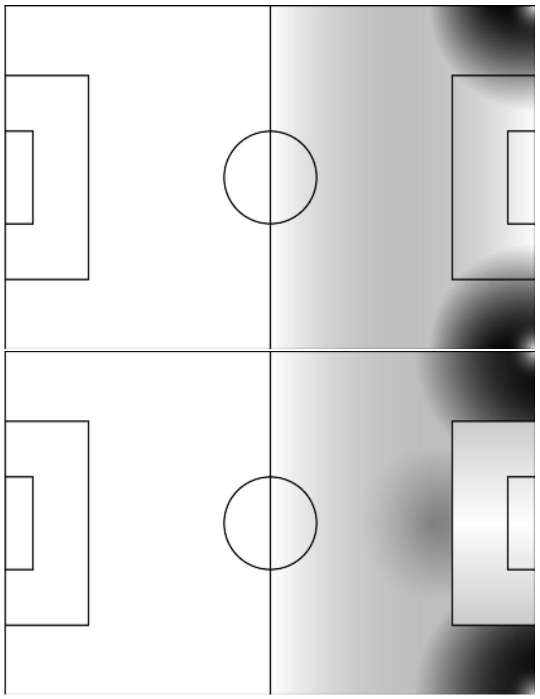
Figure 3: Sample evaluation function for the final location of the ball. Darker is a higher evaluation. The top figure is for a pass (where we control the ball) and the bottom is for a send (where the ball is kicked to no particular agent at the end of the play)

  Plans that are too long have a lower chance of succeeding and plans that are too short do not add much over reactive behaviors. Therefore, plans with length 3 (i.e. 3 passes or 2 passes and a send) have the highest value and the value degrades from there.

- Average path danger

- Maximum path danger

  Evaluation of these characteristics is implemented in two different ways. Note that high value here represents a high degree of pass safety.

  The first method samples the probability distributions of all the players in a triangle from the start of the pass to the end of the pass (see Figure 4). In order to deal with the problem that all probabilities tend to decrease as the length of the pass increases, we multiply the probability by a factor which grows as the time grows. We call this new quantity the “occupancy” of the triangle. The average and maximum values here are just 1.0 minus the average/maximum occupancy of all the passes.

  The second method uses just the means of the distributions of the opponents to estimate the probability of a pass’s success. This method (McAllester & Stone 2000) is used during the game play to evaluate many different passing options. It was designed to be run extremely quickly, improving the speed of the hillclimbing steps. However, distributional information (such as the variance)

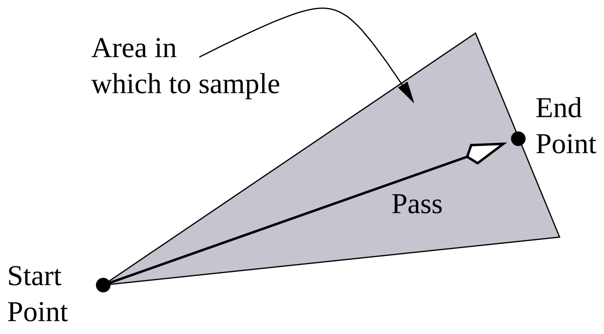

Figure 4: Sampling the Probability Distributions for Pass Danger

is ignored by only looking at the mean of the distributions. The average danger is the average pass success probability while the maximum value is actually the minimum pass probability.

The second method is much faster and was therefore used at the RoboCup2000 competition. It is still unclear if either method outperforms the other on overall plan quality however.

The opponent models (described in the next section) need to know the starting locations for the players. We use the current player locations as the opponent starting locations. Then, throughout the hillclimbing process we periodically look at where the opponents have moved and reevaluate all of the plans. There is still the problem that once our team hears the plan and begin to move, the opponents can move.

Hillclimbing has a good anytime characteristic in that as soon as our time for this step of planning is up, we have a plan ready to return (namely the best one so far). It also allows easy application of domain knowledge by giving intelligent seeds. Unfortunately, hillclimbing is also somewhat time consuming and likely will not return the optimal answer, either because of lack of time or a local maxima.

STN Generation

Once given a target path for the ball, an STN is constructed that instructs the agents in a way to make the ball follow that trajectory.

The first step is "Role Making." A separate role is created for executing each needed pass (i.e. each intersection of straight line segments). A role consists of all of the locations which the agent will need to move to and where it will kick the ball. Complication from the offsides rule[1] are handled here.

Next an STN is generated from the list of roles and locations. This is largely just a change in representation since some basic ordering constraints are present in the output of the previous step. However, upper and lower bounds are added based on parameters such as average player speed and maximum and minimum kick power. These parameters are obtained by from empirical experiments.

For the "Agent Instantiation step", each agent independently does a greedy matching of agents to roles. This is based on the (consistent) notion of the current formation of the team to and distances of home formation positions to starting role position. Formation information and consistency is obtained through the Locker Room Agreement (Stone & Veloso 1999).

[1]The offsides rule in soccer (and modelled in the Soccer Server) means that a player on one team can not be closer to the opponent goal than the last defender *when the ball is kicked*. This means that the agents must be aware of when a pass starts in order to stay onsides correctly.

Opponent Models

In order to perform the planning described in the previous section, we need a model of the opponents' movements, specifically to compute the path "danger" (in terms of the ball being intercepted by the opposing team) in the evaluation function. Because of the short time span in a simulated soccer game, we decided to fix models ahead of time and to choose between them during the game. Selecting a model should be much faster than trying to actually create a model online.

Conceptually, we want an opponent model to represent how an opponent plays defense during setplays. We expect that a wide range of decision making systems of the opponent can be roughly captured by a small set of models. In order to handle the uncertainty in the environment and to allow models to represent more information than just a simple predicted location, we use probabilistic models. Given the recent history of the ball's movement (from the start of the set play for example) and the player's initial location, the model should give, for each player, a probability distribution over locations on the field.

Let p be the number of players on a team. Let F be the set of positions on the field, discretized to 1m. We represent the ball movement as a sequence of locations on the field (an element of F^*)[2] A location for each player is a sequence of positions on the field (an element of F^p).

An opponent model defines a probability distribution for each player over the end locations on the field for that player, given the ball movement $B \in F^*$ and a set of starting positions $S \in F^p$ for all the players. If R is a probability distribution over locations on the field, an opponent model M is a function:

$$M: \underbrace{F^*}_{\text{ball movement}} \times \underbrace{F^p}_{\text{initial pos.}} \to \underbrace{R^p}_{\text{probability distribution for each player}} \quad (1)$$

In particular, for an opponent model M, the probability for player i being at end location ℓ_i:

$$P_i[\ell_i | B, S] \quad (2)$$

is calculated by choosing the ith probability distribution output by the model and calculating the probability of the location ℓ_i according to that distribution.

[2]We will use the notation F^i to denote $F \times \cdots \times F$ (F repeated i times); in other words, a sequence of i elements of F. F^* will denote $\cup_{i \in \mathbb{N}} F^i$.

Notice that each player's distribution may depend on the starting positions of *all* the opponent players. In order to avoid having to recursively model what the opponents are modeling about our movements and our locations, we do not allow the opponents distributions to depend on our own player locations. This greatly simplifies the planning described in the above section. The ball movement B is the planned ball movement. The starting positions for the players are their current locations. We then have a probability distribution for each player, which is the probabilistic obstacle for the path planning.

Model Selection

Given a set of opponent models, it is still a challenge to decide which model best describes the opponent. In other words, which opponent model should we use in order to get the best plans from our planner? We assume that the opponent has chosen one of our models at the beginning of the game and is then independently generating observations from that model. We can then use a naive Bayes classifier.

We maintain a probability distribution over the models. The original distribution (the prior) is set by hand. Then, whenever a planning stage is entered, the model with the highest probability is used. Upon observing a plan execution, we use observations of that execution to update our probability distribution over the models.

We start with a probability distribution over the set of models $\{M_1, \ldots, M_m\}$ (known as the *prior*) and then observe. An observation is a triple of starting locations for all the players $S \in F^p$, a ball movement $B \in F^*$, and ending locations for all of the players $E \in F^p$. We want to use that observation to calculate a new probability distribution, the *posterior*. That distribution then becomes the prior for the next observation update.

Consider one updating cycle of an observation $\omega = (S, B, E)$. We want $P[M_i|\omega]$. Using Bayes' rule we get

$$P[M_i|\omega] = \frac{P[\omega|M_i]P[M_i]}{P[\omega]} \tag{3}$$

We make the following assumptions, in order to simplify equation 3.

1. The players movements are independent. That is, the model may generate a probability distribution for player x based on everyone's starting locations. However, what the actual observation is for player x (assumed to be sampled from this probability distribution) is independent from the actual observations of the other players.

2. The probability of a particular starting position and ball movement are independent of the opponent model. This assumption is questionable since the ball movement (i.e. plan) generated depends on the opponent model.

$$\begin{aligned}
P[M_i|\omega] &= \frac{P[S,B,E|M_i]}{P[\omega]} P[M_i] \quad \text{(from eq. 3)} \\
&= \frac{P[E|S,B,M_i]P[S,B|M_i]}{P[\omega]} P[M_i] \\
&= P[E|S,B,M_i]\frac{P[S,B]}{P[\omega]} P[M_i] \quad \text{(assump. 2)} \\
&= \underbrace{P[\ell_1|S,B,M_i]P[\ell_2|S,B,M_i]\ldots P[\ell_p|S,B,M_i]}_{\text{what opponent model calculates (eq. 2)}} \\
&\quad \underbrace{\frac{P[S,B]}{P[\omega]}}_{\text{norm. constant}} \underbrace{P[M_i]}_{\text{prior}} \quad \text{(assump. 1)}
\end{aligned}$$

The term labeled "norm. constant" is a normalization constant. That is, it does not depend on which model is being updated, so we don't have to explicitly calculate those terms. We calculate the remaining terms and then normalize the result to a probability distribution.

Of course, the opponent hasn't actually chosen one of our models from which to generate observations. However, if any model approximates their behavior closely, that model should still be the most likely model.

One problem in this approach is that probabilities for models get arbitrarily close to zero. Because of limited numerical representation, these probabilities may get rounded to exactly zero in the updating. This can especially be a problem if an opponent behaves like one model for a while and then changes to another.

We use a form of weight sharing to handle this problem. At the end of every update cycle, a small probability mass (0.1 in our case) is added to every model and then the distribution is renormalized. This prevents any model's probability from going to 0, while not changing which model is most likely on any one update.

Empirical Results

We created several opponents models to use for testing. In all of the models used, the distribution of each players final position is represented by a 2-dimensional Gaussian with equal variance in all directions. The standard deviation is an affine function of time (since the beginning of the setplay). The mean is computed as discussed below.

We used five models for the empirical evaluation. Naturally, the mean of each player's final distribution is computed relative to the initial position as follows:

No Movement At the initial position of the player

All to Ball Moved towards the ball at a constant speed

All Defensive Moved towards the defensive side of the field at a constant speed

All Offensive Moved towards the offensive end of the field at a constant speed

One to Ball This model is slightly different from the others. The ball's movement is broken down into cycles. At each cycle, whichever player is closest to the ball is moved slightly closer. Note that since the ball can move faster than the players, which player is closest to the ball can change several times during the ball's movement. The final positions of the players are the means of the distributions.

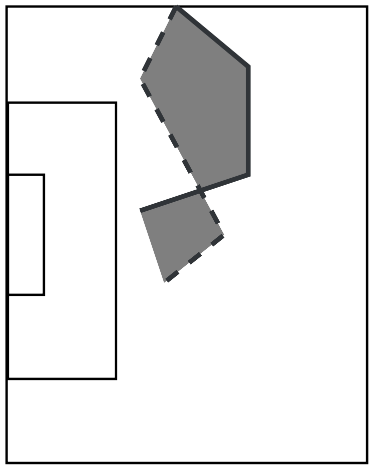

Figure 5: Example of the area between paths

Plan Set	Median Area Difference	Number of Comparisons
Planning seeds	315	138
Across opponent models	138	223
Within one opponent model	0	6625
Within one opponent model (unique plans)	116	3073

Table 1: Median area differences among several sets of plans. The area difference roughly captures the variation among the plans.

In order to evaluate the differences in the plans produced using opponent models, we compare paths by looking at the area between them. For example, in Figure 5, the shaded area is the area between the solid and dotted paths. We use this area because it expresses in simple terms how different two paths are. Hence the median area difference between a set of plans expresses roughly how much variation there is in a set of plans.

First we look at the variation in our planning seeds. This gives some idea of the maximum range we could expect plans to vary. As shown in Table 1, the median area difference in 315. Then, we compare the plans generated when the only variation is which opponent model is used for planning. Using a different opponent model for planning gives a median area difference of 138. Not surprisingly, this is somewhat lower than the median difference between the planning seeds. The evaluation of a path depends strongly upon the starting positions of the agents. The seeds are designed to roughly cover all possible starting positions of the opponents, so the variation of the hillclimbing seeds will be higher.

The variation of the plans observed by using two different models for planning is also higher than the variation observed for using just a single model. We ran our system on a set of problems 53 times in order to understand the variation in the plan returned by the hillclimbing approach. Not surprisingly, the median area is 0, since the same plan is returned many times. If we restrict our attention to just the set of unique plans returned over these 53 trials, the median area of 116 is still smaller than the median area between plans returned by different models. This suggests that, as expected, the model used causes more variation in the output of the planner than simply results from random variation in the hillclimbing.

Related Work

A great deal of work has been done in the area of plan recognition. For example, Tambe(1996) has explored tracking the high-level intentions of agent teams. This can be useful to infer events which are not directly observable by an agent. However, his work requires knowledge of the workings of the other agent through operator hierarchies. We would like to be able to relax that requirement for behavior recognition.

Other such as Charniak and Goldman(1993) have explicitly looked at dealing with uncertainty in the observations for plan recognition. They use Bayesian networks to maintain probability distributions over possible plans which could have generated the observed behavior.

Devaney and Ram (Devaney & Ram 1998) have worked on identifying behaviors in the sort of two dimensional spatial environment which we use here. In particular they looked at the movement of military troops during battle to identify behaviors through a combination of "plan recognition, pattern recognition, and object tracking." However, they are more focused on the problem of identifying particular repeated patterns of movement among the large amount of given movement data rather than trying to select a particular model which captures all of the agents' movements.

Perhaps the most similar work to ATAC is by Intille and Bobick (1999). They give a framework for recognizing plays in American football. The plays are given in terms for goals for the agents and limited temporal constraints between actions. Further, similar to (Charniak & Goldman 1993), they use Bayesian style networks to incorporate observed information.

One important difference between ATAC and other plan and behavior recognition work is that we are specifically focusing on using the recognized information for purposes for predicting and adapting to future behavior. We assume that the opponents will behave similarly to how they performed in the past (in terms of our models), and use that information to develop a plan.

Another area of related work is in plan representation. Doyle, Atkinson, & Doshi(1986) have examined inserting perceptual expectations into plans based on preconditions and post-conditions. We attach similar conditions to each parameterized action or event of the agents.

Temporal constraints have been used by Intille and Bobick(1999) to help in plan recognition. However, the temporal representation we use here is much richer and is used in execution as well as in plan monitoring.

Currently, we only detect plan failure when preparing to execute a specific action or when a temporal constraint is violated. Reece and Tate(1994) have worked on how to add monitors to plan execution to allow for earlier detection of execution problems. This would be a useful addition to the STN representation here.

Conclusion and Future Work

We have presented the ATAC system for team coaching which is adaptive to the current adversary. The main contributions of this work are:

1. We present a Simple Temporal Network(Dechter, Meiri, & Pearl 1991) based plan representation and execution algorithm for multi-agent plans. The plan representation expresses temporal coordination and monitoring in a distributed fashion.

2. We give an algorithm for generating a multi-agent plan for agent movements in the given plan representation.

3. We present a method for adaptation to adversarial strategies based on a naive Bayes classifier over fixed opponent models. This method could potentially be applied to any case where a reasonable range of probabilistic models can be determined before interacting with the adversary.

In the games at RoboCup2000, it was evident that our team benefited from adaptive setplays. Our ATAC approach created a variety of setplay plans in adaptation to completely unknown opponent teams. More targeted empirical comparison on the impact of the technique is challenging but it is part of our ongoing work. We have just completed an set of extensive empirical experiments of soccer coaching techniques which demonstrate that the set play planning described here is an important component a full set of coach abilities (Riley, Veloso, & Kaminka 2002).

During execution, the agents do not take advantage of opportunities which may occur. For example if an agent ends up with a good shot on the goal, but the plan is for it to pass, it will pass the ball anyway. Storing alternate plans and intelligently adding monitors for these plans as in (Veloso, Pollack, & Cox 1998) could make the plan execution usefully opportunistic.

Simple Temporal Networks is a promising basis for multi-agent plans. While the the representation richly expresses temporal constraints, it could benefit from a more explicit synchronization representation in terms of the observations expected for particular nodes.

In this work, the opponent models were written by hand, but all of the algorithm here could work with models which are learned by observation of opponents. We plan to continue to explore the area of generating and using opponent models for fast recognition of behaviors, in order to further exhibit adaptive, intelligent responses in our agents.

Acknowledgments

The authors would like to thank all those at Draper Laboratory who provided advice and insight. We would especially like to thank Michael Cleary for many helpful comments and being a sounding-board for most of the ideas presented here. We would also like to thank Deb Dasgupta at MIT for implementing parts of ATAC, as well as other pieces of the full simulated robotic soccer team. Lastly, but in no way least, we would like to thank Peter Stone, with whom our long collaboration in the robotic soccer domain has been both enjoyable and fruitful.

This work was largely undertaken at and supported by Draper Laboratory during the summer of 2000, while Patrick Riley was working there.

This research was sponsored in part by the United States Air Force under Cooperative Agreement No. F30602-98-2-0135. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force, or the US Government.

This material is based partly upon work supported under a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Charniak, E., and Goldman, R. 1993. A Bayesian model of plan recognition. *Artificial Intelligence* 64(1):53–79.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Devaney, M., and Ram, A. 1998. Needles in a haystack: Plan recognition in large spatial domains involving multiple agents. In *AAAI-98*, 942–947.

Doyle, R.; Atkinson, D.; and Doshi, R. 1986. Generating perception requests and expectations to verify the executions of plans. In *AAAI-86*.

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.; Tambe, M.; Stone, P.; Veloso, M.; Coradeschi, S.; Osawa, E.; Matsubara, H.; Noda, I.; and Asada, M. 1997. The robocup synthetic agent challenge. In *IJCAI-97*, 24–49.

Latombe, J.-C. 1991. *Robot Motion Planning*. Kluwer Academic Publishers.

McAllester, D., and Stone, P. 2000. Keeping the ball from CMUnited-99. In *Proceedings of RoboCup-2000*.

Morris, P., and Muscettola, N. 2000. Execution of temporal plans with uncertainty. In *AAAI-2000*, 491–496. AAAI Press/The MIT Press.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *KR-98*.

Noda, I.; Matsubara, H.; Hiraki, K.; and Frank, I. 1998. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence* 12:233–250.

Reece, G., and Tate, A. 1994. Synthesizing protection monitors from casual structure. In *AIPS-94*.

Riley, P.; Veloso, M.; and Kaminka, G. 2002. Towards any-team coaching in adversarial domains. (under review for AAMAS2002).

RoboCup Federation, `http://sserver.sourceforge.net/`. 2001. *Soccer Server Manual*.

Stentz, A. 1994. Optimal and efficient path planning for partially known environments. In *Proceedings of IEEE International Conference on Robotics and Automation*.

Stone, P., and Veloso, M. 1999. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence* 110(2):241–273.

Stone, P.; Riley, P.; and Veloso, M. 2000. The CMUnited-99 champion simulator team. In Veloso; Pagello; and Kitano., eds., *RoboCup-99: Robot Soccer World Cup III*. Berlin: Springer. 35–48.

Stone, P.; Veloso, M.; and Riley, P. 1999. The CMUnited-98 champion simulator team. In Asada, and Kitano., eds., *RoboCup-98: Robot Soccer World Cup II*. Springer. 61–76.

Stone, P. 2000. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. Intelligent Robotics and Autonomous Agents. MIT Press.

Tambe, M. 1996. Tracking dynamic team activity. In *AAAI-96*. AAAI Press.

Veloso, M.; Stone, P.; Han, K.; and Achim, S. 1998. CMUnited: A team of robotic soccer agents collaborating in an adversarial environment. In Kitano, H., ed., *RoboCup-97: The First Robot World Cup Soccer Games and Conferences*. Berlin: Springer Verlag. 242–256.

Veloso, M.; Bowling, M.; and Stone, P. 1999. Anticipation as a key for collaboration in a team of agents: A case study in robotic soccer. In *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839.

Veloso, M.; Pollack, M.; and Cox, M. 1998. Rationale-based monitoring for planning in dynamic environments. In *AIPS-98*.