# Coach Planning with Opponent Models for Distributed Execution

Patrick Riley (pfr@cs.cmu.edu) and Manuela Veloso (mmv@cs.cmu.edu)
*Computer Science Department, Carnegie Mellon University*

**Abstract.** In multi-agent domains, the generation and coordinated execution of plans in the presence of adversaries is a significant challenge. In our research, a special "coach" agent works with a team of distributed agents. The coach has a global view of the world, but has no actions other than occasionally communicating with the team over a limited bandwidth channel. Our coach is given a set of predefined opponent models which predict future states of the world caused by the opponents' actions. The coach observes the world state changes resulting from the execution of its team and opponents and selects the best matched opponent model based on its observations. The coach uses the recognized opponent model to predict the behavior of the opponent. Upon opportunities to communicate, the coach generates a plan for the team, using the predictions of the opponent model. The centralized coach generates a plan for distributed execution. We introduce: (i) the probabilistic representation and recognition algorithm for the opponent models; (ii) a multi-agent plan representation, Multi-Agent Simple Temporal Networks; and (iii) a plan execution algorithm that allows the robust distributed execution in the presence of noisy perception and actions. The complete approach is implemented in a complex simulated robot soccer environment. We present the contributions as developed in this domain, carefully highlighting their generality along with a series of experiments validating the effectiveness of our coach approach.

**Keywords:** planning, distributed execution, opponent modeling, advice, simulated robot soccer

**Abbreviations:** STN – Simple Temporal Network; MASTN – Multi-Agent Simple Temporal Network

## 1. Introduction

Multi-agent domains can include teammates and adversarial agents. One of the main challenges of such domains is the coordination and response of teammates to the adversarial agents. Often, there is no dominant strategy with respect to opponents, i.e., a team of agents can not select a strategy that has good performance independent of the adversary. Rather, the team needs to adapt its behavior by observing the adversary online.

We consider instances of *Periodic Team Synchronization* domains (Stone and Veloso, 1999), where team agents can periodically synchronize their team strategies. In such domains, a team can construct a plan

of action in a centralized fashion during a synchronization period. The plan then needs to be sent to the team for fully distributed execution. Coordinated execution faces the challenges of noisy and incomplete sensations, noisy actions, and limited and unreliable communication.

This article reports our work on addressing this online team adaptation to an adversary. We use a coach agent that acts as the centralized planning agent. The coach has a global view of the world, does not directly act in the world, and has occasional communication with its teammates. The main steps of our approach are:

- The coach agent is equipped with a number of pre-defined opponent models as probabilistic representations of predicted states of the opponents. The coach gathers global observations that include its teammates' and opponents' states. The coach selects the opponent model that best matches the coach's observations.

- At synchronization opportunities, the coach creates a team plan that is a function of the selected model of the opponents' behavior. The plan is generated by a hill-climbing search in plan space. The evaluation function embeds the predictions of the opponent model.

- The plan is encoded in a multiagent plan representation, a Multi-Agent Simple Temporal Network (MASTN), which is a refinement of a Simple Temporal Network (Dechter et al., 1991). The MASTN representation effectively captures: (i) the different agents responsible for each plan step; (ii) the temporal dependencies between the plan steps; and (iii) bounds on the expected execution times of the actions.

- While the coach is a centralized planning agent, the agents must execute the plan in a fully distributed manner with noisy, incomplete views of the world state. The coach observes the execution of the plan in order to refine the selection of an opponent model for future plans.

We use simulated robot soccer as a rich multi-agent environment including fully distributed agents in two different teams of up to eleven agents, plus a coach agent for each team (Noda et al., 1998). Agents can reactively respond to the opponents' positioning, for example, by explicitly moving to open field positions (e.g., Veloso et al., Stone et al., 1999, 2000). Our work aims at modeling the opponent to *predict* their behavior, and deliberatively plan in response to the predicted actions.

We specifically focus on situations where the game is stopped. At these times, a team can coordinate and commit to a plan of action

to execute once the ball is back in play. These plans are known as *setplays*. Several preset setplay plans have been introduced that provide opportunities to position the teammates strategically and have been shown to contribute to the performance of a team (Stone et al., 2000, Stone et al., 1999, Veloso et al., 1998b). We contribute adaptive setplays that are created online by the coach agent.

The MASTN representation and execution algorithm are explicitly put forth in non-soccer specific terms. The plan generation algorithm uses an evaluation function based on specific soccer knowledge, but the general hill-climbing strategy should be applicable elsewhere.

A coach using the techniques presented in this article was part of a team that competed in the RoboCup competitions in 2000, 2001, 2003, and 2004. The coach created a variety of setplay plans, adaptively responding to completely unknown opponent teams. The design of the coach competition, inevitably carried out in limited time, did not allow for the gathering of statistically significant results.[1] We present controlled empirical results explaining and demonstrating the effectiveness of our approach.

In summary, the contributions of this work are:

- A coach that plans for a team of agents in a continuous environment based on online recognition of a model of opponent agent behavior, given a predefined set of opponent models.

- A plan representation and an execution algorithm for distributed execution that use temporal constraints to express coordination and facilitate failure detection. The representation and execution algorithm are general beyond the robot soccer domain.

- An empirical validation of the fully implemented system in a complex simulated robot soccer domain.

The article is organized as follows. Section 2 briefly describes the simulated robot soccer environment used as a testbed for this work. Section 3 describes our novel multi-agent plan representation and the algorithm used to execute plans. Section 4 describes how our coach agent generates plans using models of the movements of the opponent team, which are described in Section 5. Section 6 presents several empirical results and Section 7 presents related work. Finally, Section 8 concludes and gives directions for future work.

[1] Our coach placed first in 2001 and Riley et al. (2002) provides a systematic and in-depth set of experiments analyzing and validating this result. Kuhlmann et al. (2005) provide a similar set of experiments for 2003.

## 2. The Environment

All agents described in this article are fully implemented in the Soccer Server System (Noda et al., 1998) as used in the RoboCup research initiative (Kitano et al., 1997). The Soccer Server System is a server-client system that simulates a soccer game between distributed agents. Clients communicate using a standard network protocol with well-defined actions. The server keeps track of the current state of the objects in the world, executes the actions requested by the clients, 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. Figure 1 shows a screenshot.

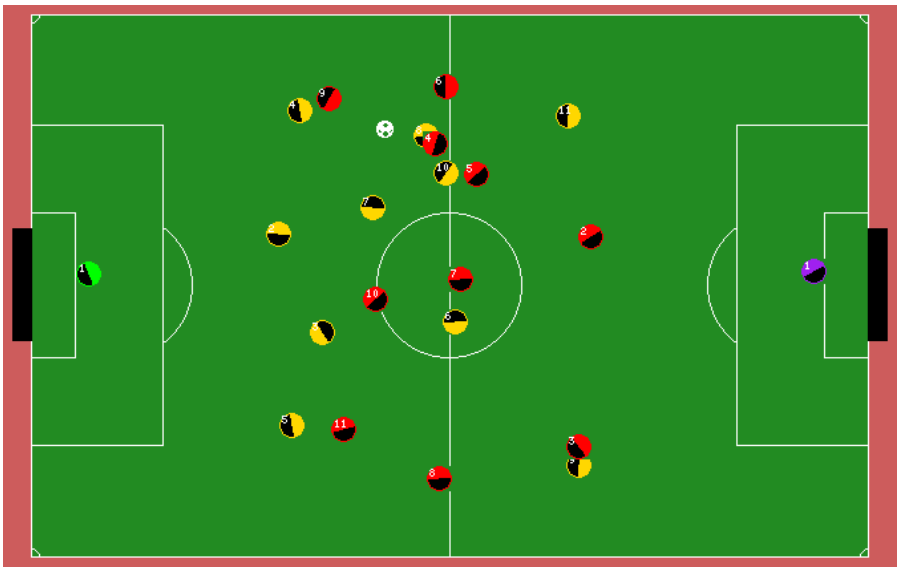

*Figure 1.* Screen shot of the Soccer Server System. The agents are represented as circles, with the light colored portion indicating which direction they are facing. Agents on the two teams are distinguished by the colors, with the light colored agents on one team and the dark colored agents on the other. The ball is the white object above and to the left of center. The coach agents are not depicted.

The agents must communicate with the server at the level of parameterized actions like turn, dash, and kick. Higher level actions, such as passing and going to a position on the field, must be implemented by combining the lower-level actions that the server understands. For example, moving to a location requires a combination of turning and dashing actions, and kicking the ball in a direction is usually a combination of a number of small kicks and player movements to accelerate the ball in the correct direction. This process, in combination with

the perception and action noise of the environment, results in noisy execution of the higher level actions. In particular, there is noise in the exact result of the action and the time taken to execute it.

There are eleven 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. The coach does not get information about the percepts received by the players or the actions that they have tried to execute.

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.

The agents can communicate, but the communication is of limited bandwidth and unreliable. Furthermore, only agents within 50m (approximately half the length of the field) of 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 regardless of their location. However, the coach is allowed only occasional communication with the players, namely when the play is stopped (due to an out of bounds call, kick-off, etc.) or every 30 seconds, whichever is sooner. The coach's utterances can not be heard by the opponent.

A few additional points should be noted:

- The world can change very quickly. Based on the maximum agent speeds and size of the field, we calculate that in about 15 seconds, the world could transition to almost any state.[2]

- The score and global clock are the only shared state features. For all other information, the agents must rely on their own local view and communication from teammates. Because of noise in sensations, the agents' beliefs about the locations of objects in the world can be inconsistent.

- There is an active and intelligent opponent team.

We used server versions 6.07 and 7.10 for the experiments. Technical details about the Soccer Server System can be found at Chen et al. (2001).

[2] Agents can move at about 10m/second and the field diagonal is about 123m, plus some time to manipulate the ball if necessary.

## 3. Multi-Agent Plans

In this article, we contribute a multi-agent plan representation and execution algorithm for distributed agents. This section covers these in detail.

Our coach agent generates movement plans for its teammates and the ball. The coach is a centralized planner, but the execution must be done in a fully distributed fashion. Therefore, the coach must encode sufficient information into the plan to allow the agents to coordinate and identify failures during execution.

The domain-independent portions of the plan representation and execution process are described in this section, as well as their use in the simulated robot soccer environment.

In order to clarify the discussions of simulated robot soccer, we first illustrate what an execution of a setplay plan looks like. Figure 2 shows the movements of the ball and players over time. Teammate 1 starts with possession of the ball. Teammate 1 passes to teammate 2, which moves to receive it. Simultaneously, teammate 3 moves forward. Teammate 2 then passes to teammate 3. Note that agent actions are going on in parallel and that some actions (like the passes) require coordinated efforts of more than one agent. The plan in Figure 2 will be used to illustrate the plan representation and execution algorithm in the following sections.

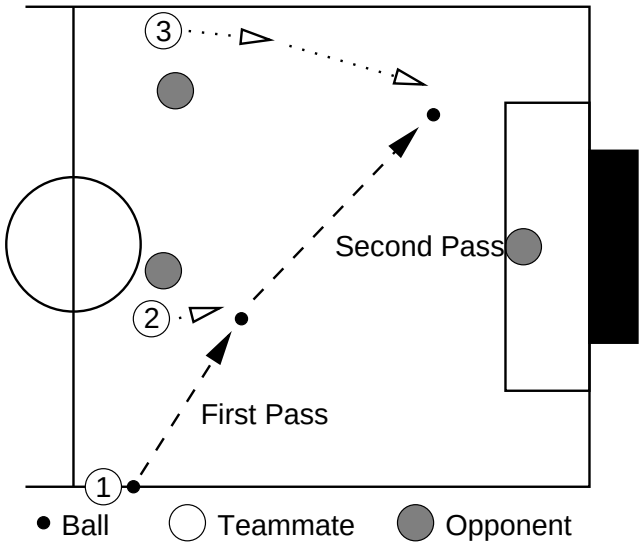


*Figure 2.* An example plan. The arrows indicate movement of the ball or a player.

The following notation will be used in the remainder of the article. Sets will be typeset in script, e.g. $\mathcal{E}$, $\mathcal{N}$, $\mathcal{X}$. The notation $\mathcal{F}^i$ is used

to denote $\mathcal{F} \times \ldots \times \mathcal{F}$, the set of sequences of $i$ elements of $\mathcal{F}$. $\mathcal{F}^*$ will denote $\cup_{i \in \mathbb{N}} \mathcal{F}^i$. Functions will be typeset in capital letters and in italics, e.g. $F$, $L$. For elements of a set given by Cartesian products, [] will be used to specify components. For example, if $x \in \mathcal{A} \times \mathcal{B}$, then $x[1]$ is the $\mathcal{A}$ component and $x[2]$ is the $\mathcal{B}$ component.

### 3.1. Plan Representation: MASTN

We introduce Multi-Agent Simple Temporal Networks (MASTNs) as a plan representation for distributed execution. MASTNs are a refinement of Simple Temporal Networks as introduced by Dechter et al. (1991). The refinements allow us to the define the execution algorithm discussed in Section 3.3. Taken together, the representation and execution algorithm are a novel scientific contribution.

We will first present Simple Temporal Networks, then introduce our refinements to make Multi-Agent Simple Temporal Networks. Our formal presentation is a combination of the notations of Dechter et al. (1991) and Morris and Muscettola (2000).

A Simple Temporal Network is a tuple $\langle \mathcal{V}, \mathcal{E}, L \rangle$ where

$\mathcal{V}$ is a set of nodes. Each node represents an event that occurs at a point in time. Activities with duration must be broken up into two events, one representing the beginning and one representing the end of the activity.

$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of directed edges between the nodes. Each edge represents a temporal constraint between nodes.

$L$: $\mathcal{E} \rightarrow (\mathbb{R} \cup -\infty) \times (\mathbb{R} \cup \infty)$ is the labeling function on the edges. The label represents a temporal constraint between the events. For example, if $\langle a, b \rangle \in \mathcal{E}$ and $L(a, b) = \langle x, y \rangle$, then event $b$ is constrained to occur at least $x$ time units after $a$, but no more than $y$. Note that $x$ and $y$ could be negative (indicating that $b$ should occur before $a$) or negative or positive infinity respectively.

STNs are a representation of a subclass of temporal constraint satisfaction problems. A user of an STN often wants to answer questions like "Is there any set of times at which my events can occur such that no constraints are violated?" or "At what time should each event be executed so that no constraint is violated?" Unlike more general temporal constraint satisfaction problems, these questions can be answered in polynomial time for STNs.

We now define Multi-Agent Simple Temporal Networks (MASTNs). We assume we are given the following:

$\mathcal{A}$ A set of identifiers for the agents.

$\mathcal{T}$ A set of node types. Nodes represent events that are brought about by the agents, so this set is similar to the set of actions available in traditional planning problems. The nodes can be parameterized. For example, in the classic blocks world there would be node types for "PickUp(RedBlock)", "PickUp(GreenBlock)", etc. Node types do *not* include information on which agent performs the event. Agents will be associated with nodes in the representation of a particular plan.

We then define the set of all possible nodes (i.e. events) $\mathcal{N}$ as $\mathcal{T} \times \mathcal{P}(\mathcal{A}) \times \mathbb{N}^*$. The $\mathcal{P}(\mathcal{A})$ represents the set of agents responsible for bringing about this event. The $\mathbb{N}^*$ element is used to provide *pointers* to other nodes (described below). The pointers allow some information to be specified in only one plan node rather than being duplicated.

An MASTN plan is then a tuple $\langle \mathcal{V}, \mathcal{E}, L, O \rangle$. These elements are:

$\mathcal{V} \subseteq \mathcal{N} = \mathcal{T} \times \mathcal{P}(\mathcal{A}) \times \mathbb{N}^*$ The set of nodes of the plan. Note that each node contains more information than in a normal STN, which considers each node as a black box.

$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ The set of edges, just as in an STN.

$L$: $\mathcal{E} \rightarrow (\mathbb{R} \cup -\infty) \times (\mathbb{R} \cup \infty)$ The labeling function on the edges, with the same meaning as in an STN.

$O$: $\mathcal{V} \rightarrow \mathbb{N}$ An ordering function on the nodes. In other words, for $v_1, v_2 \in \mathcal{V}$, $O(v_1) < O(v_2)$ means that $v_1$ comes before $v_2$. This ordering function is used as the address space of the node pointers and to allow the agent to break ties when the temporal constraints allow multiple events to be executed. The details of this tie-breaking are discussed in Section 3.3. $O$ is *not* a full temporal ordering of the nodes.

The authors are not aware of any other work where STNs are used as a basis for a multi-agent plan representation for distributed execution. In particular, by "distributed execution" we mean that each agent maintains its own perception of the current state of plan; there is no assumption of shared global sensations or plan state. Temporal networks have several properties which are useful in a multi-agent context:

- The network 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. Constraints can naturally catch violations of limits on the extent of events or if an agent fails to take appropriate action at the correct time.

- 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. If an agent is responsible for executing $e$, the agent can determine where in the world to look to observe the future execution of the event which is preventing $e$ from being ready.

We introduce MASTNs as a way to take advantage of these properties for multi-agent plan execution.

### 3.2. MASTNs in Robot Soccer

This section demonstrates how the MASTN representation is applied to simulated robot soccer. We have to instantiate the domain specific values $\mathcal{A}$ and $\mathcal{T}$. Let $\mathcal{L}$ be the set of locations on the field.

- $\mathcal{A}$ is the set of numbers 1 through 11. Each agent is identified with a unique number.

- $\mathcal{T}$ is the set of node types, some of which have parameters. For example, if a node type $t$ has a parameter of type $\mathcal{L}$, then $\forall x \in \mathcal{L}, \langle t, x \rangle \in \mathcal{T}$. In particular, this means that all parameters are bound and fixed for a plan; the values of parameters to nodes do not change during plan execution.

  **Initial Position** This node represents the event of all agents arriving at their initial position for the player and the play starting. This node has a parameter which is an element of the set of partial functions from $\mathcal{A} \rightarrow \mathcal{L}$, mapping agents to their initial locations. This node is also the root node for execution, called "the beginning of the world" by Dechter et al. (1991).

  **Start Goto** This node represents the beginning of the activity of a given agent going to a location on the field. This node takes a parameter (which is an element of $\mathcal{L}$) indicating where to go.

  **End Goto** This node represents the conclusion of the move begun by the "Start Goto". The node pointer element is used to specify the associated Start Goto node.

**Start Pass** This node represents the beginning of a pass activity, where one agent kicks the ball to a location on the field. This node takes a parameter which is an element of $\mathcal{L}$ indicating where to pass. While another agent should receive the pass, that information is not explicitly represented in this node.

**End Pass** A pass is represented with three nodes: a Start Pass (for the kicker to start the ball), a Start Goto (for an agent to start moving to receive the pass), and an End Pass which represents the conclusion of both of those activities. The node pointers are used to specify the related nodes.

**Clear Ball** This node represents an agent kicking the ball to a location. This node takes a parameter which is an element of $\mathcal{L}$ indicating where to kick the ball. 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.

An example plan is graphically shown in Figure 3. A successful execution of this plan is depicted in Figure 2. The top three nodes represent the first pass from player 1 to player 2. The bottom two nodes represent the simultaneous movement of player 3. The last three nodes (in the middle right of the figure) represent the pass from player 2 to player 3.

### 3.3. Plan Execution

Plan execution is done in a fully distributed fashion. This means that each agent maintains its own data structures describing the state of the plan and independently decides what actions to take. However, it is assumed that every agent can get some knowledge of the events in the plan being executed (either by communication or observation), though it is *not* required that agents agree exactly on the timing and sequencing of events. In domains with limited observability or latency in the communication between agents, exact agreement is difficult to obtain. Therefore, flexibility in agent agreement is important. In the simulated soccer environment, the agents do have a shared global clock. In general, however, the agents do not have to agree on the exact time as long as each one can measure the progress of time accurately. In other words, as long as each agent can measure how long a second is accurately, it does not matter if all their watches agree.

Throughout, we will talk about "executing" a node. This means that the agent is taking individual actions to accomplish the event represented by the node. As discussed in Section 2, an agent is not able

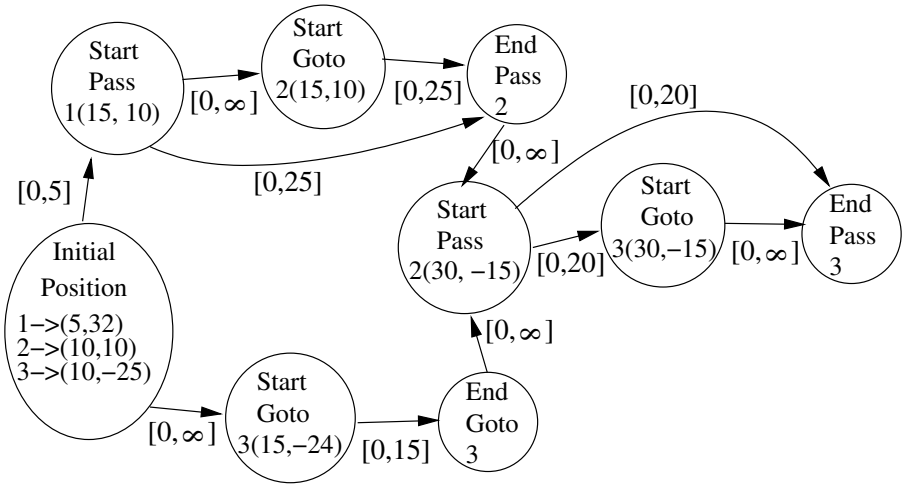


*Figure 3.* An example MASTN. The nodes represent events and the edges represent temporal constraints. The agent responsible for bringing about the occurrence of the event is shown inside each node. The numbers in parentheses are vectors representing locations on the field (elements of $\mathcal{L}$). These vectors are the parameters to the various node types. The node pointers are *not* explicitly represented in this picture. This plan corresponds to the execution depicted in Figure 2.

to precisely control when an event will occur, even when the agent is the one executing the node. Further, we assume that each agent can only execute one node at a time. That is, all of the parallelism in the plan takes place by multiple agents performing actions simultaneously.

In addition to executing various behaviors, the agents will be monitoring the execution of actions that other agents are supposed to perform. If an agent is slow or failing to execute its action, any agents that are temporally constrained by that undone action will wait for it to be done. Note that currently the agents do *not* take over actions if the currently assigned agent fails to execute it.

Muscettola et al. (1998) have described a "dispatching execution" algorithm for STN execution. This method 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. We then use the dispatching execution algorithms as subroutines to propagate temporal constraints and identify violations.

Figure 4 shows the full execution algorithm. The algorithm calls some domain specific functions whose purpose will be discussed as the algorithm is covered in detail.

```
1  initialize(p: plan)
2       ∀n ∈ p.V
3            exectime(n) = ∅
4            window(n) = ⟨−∞,∞⟩
5            construct all-pairs network for p
6  execute(p: plan, t: time, a: agent)
7       if global-monitor() = fail
8            return abort
9       ∀n ∈ p.V
10           if (exectime(n) = ∅ and node-completed(n))
11                exectime(n) = t
12      propagate-constraints()
13      ∀n ∈ p.V
14           if (constraints-violated)
15                return abort
16      // F is the set of nodes to still execute for this agent
17      F = {n ∈ p.V | exectime(n) = ∅ and a ∈ n.agents}
18      if (F = ∅)
19           return plan-completed
20      // mynode is the next node for this agent to execute
21      mynode = argmin_{n∈F} p.O(n)
22      if (not node-precondition(mynode))
23           return abort
24      if (t < window(mynode)[1])
25           holdingnodes = {n ∈ V | exectime(n)=∅ and
26                                   ⟨n, mynode⟩ ∈ p.E and
27                                   p.L(n, mynode)[1] ≥ 0 }
28           if (holdingnodes = ∅)
29                return in-progress // agent waits for time to pass
30           else
31                look-at(holdingnodes)
32                return in-progress
33      execute-node(mynode)
34      return in-progress
```

*Figure 4.* Plan execution algorithm for an individual agent. "exectime" and "window" are data structures maintained during execution. Functions *in italics* must be provided by the user of this algorithm.

The "initialize" function initializes the data structures for the given plan. In particular, the temporal constraints are set up for the dispatching execution algorithm. Then, at each time step, each agent should run the "execute" function. The execution will result in some action to take for this time step. At the next time step, the "execute" function is run again. Over time, an agent will work to execute various nodes (by taking one of more actions to accomplish each one) in the plan; it may take more than one action/time step to complete execution of a given node. During a successful execution, nodes in the plan will be marked as completed as the plan progresses.

First in the plan execution function is a call to *global-monitor*. This domain specific function should implement monitoring conditions which apply to the whole plan. Returning "fail" indicates that the plan should be aborted. In the soccer environment, we have global monitoring conditions to catch situations like the ball going out of bounds or the opponent intercepting the ball.

Next, in lines 9–11, the nodes in the plan that have not been marked as executed are checked. The domain specific function *node-completed* must be provided to identify when a node has been executed. This decision can be based on perceptions or communication. In general, this decision will depend on the type of node $\mathcal{T}$ and the agents tasked with executing the node.

The functions propagate-constraints and constraints-violated are provided by the dispatching execution algorithm (Muscettola et al., 1998). The function propagate-constraints will update the "window" data structure with allowable times for the nodes to execute and constraints-violated will indicate whether any temporal constraints have been violated.

The next section of the execute function (lines 16–21) identifies the next node for the agent to execute. The ordering function determines which node should be executed next, from the set of uncompleted nodes that this agent is responsible for.

Once the next node to execute is found, preconditions of that node can be checked (lines 22–23). For example, in the soccer environment, a 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 and Stone, 2001).

If the temporal constraints do not allow the current node to execute (line 24), then there are two cases. If there are no unexecuted nodes which must execute before this one, then the agent just waits for time to pass (lines 28–29). If there is such a node, the domain specific *look-at* function is called to tell the agent to watch for the execution of that

node (lines 30–32). In the soccer environment, this is done by having the agent face the point where the execution of that node should occur. In general, this could be done with a communication request or any other observational means.

Otherwise, the agent works towards executing the next node (lines 33–34). Note that the execution of a node may take more than one step and the agent is not required to precisely control when the node executes. These criteria allow more freedom in how the execution of nodes is carried out. For the soccer environment, *execute-node* is written using the reactive CMUnited99 (Stone et al., 2000) layer to get robust performance of such commands as "get the ball" or "kick the ball hard in direction $x$."

This algorithm is run by each agent in parallel every step. Each agent maintains its own perception of the state of the plan; there is no centralized control instructing agents when to perform actions. While STNs used in scheduling tasks can provide coordination of agents, the agents usually have access to shared global state or a shared controller, neither of which we have here. We allow the agents' perceptions of the execution state to differ as long as the difference does not cause a violation of the temporal constraints.

For example, since the agents in the soccer domain use noisy, limited observations to determine when a pass has started, the agents will in general not agree on the exact time that the pass started. Noise can make the ball appear to move when it hasn't or the agent may not be looking at the ball when the pass starts. Even if none of the agents agree on the exact time, the plan execution may still be successful as long as the temporal constraints are not violated.

It is difficult to make precise statements about how much the agents' perceptions of the world are allowed to differ. The difference allowed will depend on the how much flexibility there is in the plan. Plans could be constructed such that there is exactly one time at which each event could execute, giving no flexibility in agents' perceptions. The plans generated here do allow for differences in agent perception and it is an open question how much flexibility is allowed in typical plans for the soccer domain or for other interesting domains.

### 3.4. Plan Execution Example

We will now illustrate an execution of the plan shown in Figures 2 and 3. This section will not attempt to cover all the steps of the algorithm described in Section 3.3. Rather, some of the important points relating to distributed execution will be highlighted.

Figure 5 shows each agent's perception of when the events in the plan occur. The shaded events are events for which that agent is responsible. Each agent is responsible for its position in the Initial Position event. After that, agent 3 starts going to its next position. Simultaneously, agent 1 starts the pass to agent 2 (SP1 $\rightarrow$ 2) and agent 1's role in the plan is complete. As shown by the shading, agent 1 no longer tracks the execution of the plan. Some time later, agent 2 realizes that the pass has begun and starts to go to the reception point for the pass (SG2). Meanwhile, agent 3 completes going to the intermediate point (EG3). Once the first pass is complete (EP1 $\rightarrow$ 2), agent 2 passes to agent 3 (SP2 $\rightarrow$ 3). Agent 2's role is then complete. Agent 3 then proceeds to get the ball to complete the pass (SG3 and EP2 $\rightarrow$ 3).

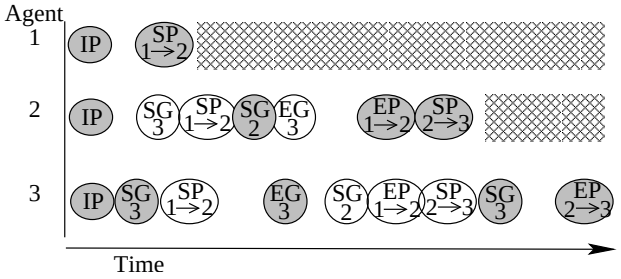


*Figure 5.* Example execution of a plan. This is an execution of the plan in Figure 3 and illustrated in Figures 2. Each agent (shown on the rows) has its own perception of when events occur; the time an agent perceives an event to occur is shown by the event's horizontal location. The shaded events in each row are the events which that agent is responsible for bringing about. IP stands for Initial Position, SP for Start Pass, EP for End Pass, SG for Start Goto, and EG for End Goto.

Note that the agents perceive events' execution times to be different and that they do not even always agree on the ordering (e.g. SG2 and EG3). However, each agent's perception of the order of events must obey the temporal constraints in the STN (Figure 3). Also, at every point, each agent will verify global and local constraints and abort the plan if the verification fails. For example, if an opponent agent intercepts the first pass, the agents will stop the plan, communicating their perception of the need for termination.

## 4. Plan Creation

Given the MASTN plan representation described in Section 3, it is still a significant challenge to generate these plans, especially accounting for the predicted behavior of the adversary. We divide the process of plan creation into four steps. The particular implementations of these

steps rely on soccer specific knowledge, but this general breakdown would likely be useful in other domains. The *Waypoint Planning* stage is described in Section 4.1 and the *Role Making*, *MASTN Compilation*, and *Agent Instantiation* steps are described in Section 4.2.

### 4.1. Waypoint Planning

In order to plan the waypoints, we use models of opponent movement. Section 5 fully describes these models, but for the purposes of this section, just consider waypoint planning as a path planning problem with straight-line segments and dynamic, probabilistic obstacles (the obstacles are the opponents). Unlike many path planning problems, the obstacles are not fixed regions in known locations. Rather, we have a probability distribution over each obstacle's locations over time.

The opponent models which describe the movement of the obstacles take into account the current positions of the opponents and the predicted actions (i.e. the waypoints) to produce the predicted movements of the opponents. The exact positions of our teammates will be determined from the waypoints, but those positions are not *explicitly* part of the opponent model. Further, the waypoint planning ignores the current positions of the teammates since it is assumed that the team can move into the starting positions before the plan begins.

The problem addressed here is significantly different from a traditional shortest-path planning problem. Selecting a path in this environment inherently involves tradeoffs. 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 tradeoff in the length and safety of the path. Further, we do not have a single goal position, but rather a ranking of the possible positions. That is, not every setplay will result in a goal being scored, but some final positions from the setplay are better than others.

These constraints make it difficult to apply many of the traditional path planning methods, such as those described in Latombe (1991). Planning methods that 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, or expect replanning to be available. Approaches that 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. We wanted our coach to come up with a complete plan, not

rely on the distributed, executing agents to replan. Replanning would be difficult in this case both because of the partial observability of the executing agents and the unreliable communication.

In order to plan in this challenging domain, we decided to directly specify an evaluation function for paths and use hillclimbing on a set of paths to find a locally optimal path. The evaluation function for the paths will include the processing of the probabilistic opponent model.

Our evaluation function meets the basic requirement of hillclimbing that nearby paths have similar evaluations. Also, our plan space is about $10^{18}$ so we can not cover a large proportion of the space in the few seconds available for planning.[3] Note that the coach can not plan until the ball actually goes out of bounds because the plan evaluation depends on the current location of the ball and opponents.

We first describe the hillclimbing algorithm and then describe the evaluation function. Our path planning algorithm is shown in Figure 6. Note that sometimes (decided by the variable $A$) we move only a single point in a hillclimbing step and sometimes we move the entire tail of a path. By varying the neighborhood considered, the hillclimbing should be able to escape more local minima.

```
S := Set of starting paths
while (there is time left)
    Uniformly randomly remove a path p from S
    Uniformly randomly pick a point x on p
    Uniformly randomly set A to true or false
    bestp = p
    ∀ small displacement vector v
        Make path p' by moving x by v
        If (A)
            In p', move all points after x by v
        If eval(p') > eval(bestp)
            bestp = p'
    Insert bestp into S
    If (time left < half of original time)
        Remove all but current best path from S
```

*Figure 6.* Hillclimbing algorithm for waypoint planning

Note that the hillclimbing runs for a fixed amount of time. By the standard simulated soccer rules, the team has 20 seconds from the time

[3] The plan space size was estimated as follows. The field was discretized to 1m. Passes were considered between 8m and 38m and sends between 38m and 55m. A plan could be up to four segments with the last possibly being a clear.

play is stopped to begin the setplay. The coach plans for 5 seconds, almost all of which is used for hillclimbing. Communicating the plan usually takes 2 to 4 messages, and a message can be sent every 100ms. Halfway through the 5 seconds of hillclimbing, 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 dependence is necessary because different types of plans make sense for different situations such as whether the ball is in the middle or side of the field, whether the ball is near our goal, whether this is a corner kick, etc. Since only a small part of the plan space can be explored by hillclimbing, setting good planning seeds can greatly help in finding a high-quality plan. Figure 7 gives an example of the hillclimbing seeds for one setplay situation.

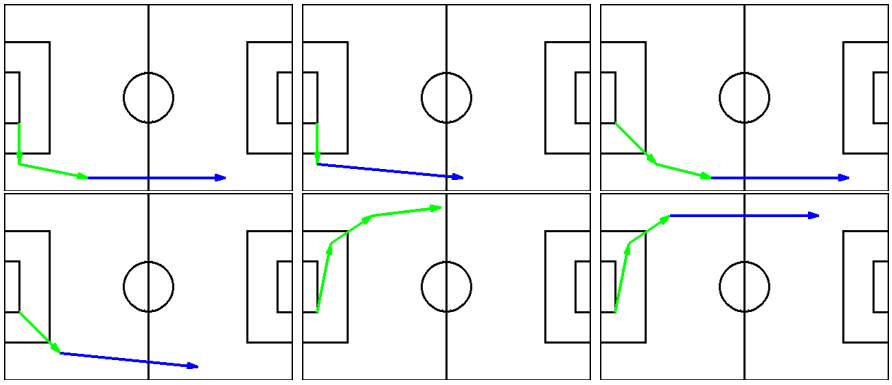

*Figure 7.* Example hillclimbing seeds. These are the seeds for a goal kick from the bottom side. The longer, darker arrows are clears and the shorter, lighter ones are passes.

The set of starting paths is a natural point for inserting domain knowledge into the system. In particular, for the soccer environment we can give the basic shapes of paths, such as passing to the outside then clearing from a free kick. If there is no domain expert to provide this knowledge, random starting paths or paths taken from past executions could be used.

The crucial part of hillclimbing is the evaluation function (*eval* in Figure 6). While the particular evaluation function chosen here is specific to the soccer domain, the general idea of hillclimbing in plan space with a domain specific evaluation is applicable to other domains. We use the following weighted factors:

– Player 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 plan is a clear (kicking the ball down field with no specific agent designated to get the ball), this has value 0.0. That is, it is better that our team ends up with control of the ball rather than just kicking it down the field.

– Ball's end location

The value of the ball's end location also depends on whether we are in control of the ball at the end of the play. In other words, the value of a position of the ball varies based on whether we have a teammate in control of it. Getting the ball near the goal and the opponent's baseline has high value, and just getting the ball further down field is also of high value. The functions are shown graphically in Figure 8. The functions for a pass and a clear are almost identical except for two things. First, the clear figure has less value inside the penalty box (if the ball is kicked into the penalty box with no one nearby, the goalie will just grab it). Second, the clear has additional value near the top of the penalty box because it may induce the goalie to move out of the goal, giving a good shot if the ball can be retrieved and passed to an agent on one of the sides. Readers interested in the exact definition of the evaluation function should see the article's online appendix.[4]

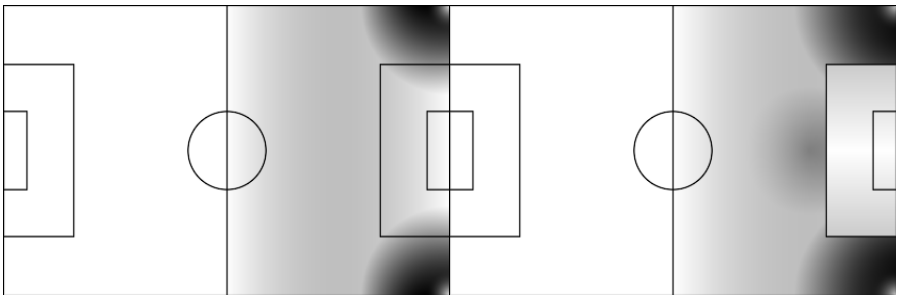

*Figure 8.* Evaluation function for the final location of the ball. Darker is a higher evaluation. The left figure is for a pass (where we control the ball) and the right is for a clear (where the ball is kicked to no particular agent at the end of the play).

– Length of plan

Since every action has some probability of failure, long plans generally have a lower chance of succeeding than shorter ones. However, short plans add less to the team behavior simply because they have

[4] `http://www.cs.cmu.edu/~pfr/appendices/2004jaamas.html`

less time to affect the behavior. This factor makes this tradeoff explicit. Therefore, plans with length 3 (i.e. 3 passes or 2 passes and a clear) have the highest value and the value degrades from there. The values here are heuristic and chosen based on experience with this simulated robot soccer domain. The exact values are shown in Table I.

Table I. Evaluation values for the "length of path" component

| Length | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Value | 0.4 | 0.8 | 1.0 | 0.5 | 0.2 |

– Average path safety and minimum path safety

  These are two different measures of the safety of the path. The safety of each segment in the path is first evaluated and then the average and minimum of those values is computed. A high value here represents greater safety.

  We use the means of the distributions of the opponents to estimate the probability of a pass's success. This method, introduced by McAllester and Stone (2001), is used during normal game play to evaluate many different passing options. It was designed to be run extremely quickly, improving the speed of the hillclimbing steps. The average safety is the average pass success probability while the minimum value is the minimum pass success probability.

The factors are then added together with the weights shown in Table II. The weights were obtained through hand tuning after the system was implemented.

Table II. Weights for combining the factors for the hillclimbing evaluation function

| Factor | Weight |
|---|---|
| Player control at end | 0.22 |
| Ball's end location | 0.2 |
| Length of path | 0.1 |
| Average path safety | 0.33 |
| Minimum path safety | 0.33 |

Hillclimbing has a good anytime characteristic in that as soon as our time for 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 maximum.

### 4.2. Waypoints to Complete Plan

Given a target path for the ball, the coach constructs an MASTN in three phases:

**Role Making** A separate role in the plan is created for executing each needed pass (i.e. each intersection of straight line segments). A role consists of all of the locations that the agent will need to move to and where it will kick the ball. This process creates the set of nodes $\mathcal{V}$ and the ordering function $O$. In particular, nodes of the correct types and their associated parameters and pointers are created. The agent values (members of $\mathcal{A}$) are temporary, to be replaced by the correct agents later in the process. Some domain specific requirements for the soccer environment, in particular for the offsides rule[5], are handled here.

**MASTN Compilation** The step adds the edges ($\mathcal{E}$) and their temporal constraint labels ($L$). Domain specific knowledge such as the speed of running and kicking and their normal variations is used to establish the time bounds for execution between the various events.

**Agent Instantiation** This step assigns specific agents to the roles in the plan. Role allocation is an important problem in multi-agent systems (e.g., Weiss, 1999). However, for this system, we use a simple domain specific algorithm. The assignment is done using the current formation of the team and a greedy matching algorithm between the agents' home positions and the plan's starting positions.

If the coach knows the current formation, the coach can perform this step. However, this can also be done by the players, as long as their formation information is consistent. For the players, formation information and consistency is obtained through the

[5] The offsides rule in soccer (and modeled 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.* For the planning, the offsides rule means that the agents must be aware of when a pass starts in order to stay onsides correctly.

Locker Room Agreement (Stone and Veloso, 1999). The Locker Room Agreement is a set of preset knowledge that allows the agents to agree on some aspects of play and strategy changes based on any shared state features (e.g. the game time).

## 5. Opponent Models

This section describes the opponent model representation and the algorithm used for selecting a model from a given set. In order to perform the planning described in Section 4, we use a model of the opponents' movements, specifically to compute the path "safety" (in terms of the ball being intercepted by the opposing team) in the evaluation function.

Throughout, we will not be reasoning about any modeling that the opponents do of our team. This choice is primarily for computational tractability and simplicity; handling recursive agent modeling can be quite challenging (Gmytrasiewicz and Durfee, 1995) and is outside the scope of this work. Also, the amount of data we have to work with about a particular opponent is quite small. If we considered richer opponent models involving recursive modeling, we may need more data to correctly recognize the models.

Because of the short time span of a simulated soccer game, we decided to begin a game with a fixed set of models and choose between them during the game. Selecting a model should require fewer observations than trying to create a model from scratch.

The opponent models we present will specifically model opponent movement and position in a two dimensional plane, though the representation and algorithms should easily extend to higher-dimensional metric spaces. We use the models to evaluate the quality of various possible planned actions, but the same models would be useful for any application where prediction of opponent movement would be useful.

These models are not intended to capture the full strategy or movements of the opponents. Rather, the models only need to capture the way the opponents move in the two to twenty seconds after the game has stopped, while the setplay is actually going on.

Two assumptions related to the opponents should be noted. First, for best effectiveness, these models should have good predictive power with respect to the set of opponents expected. We assume that the variation in opponents can be approximately expressed in a reasonably sized set of models from which to choose.

Second, the output of an opponent model does not *explicitly* depend upon the positions of our players. However, the output does depend on

the anticipated path of the ball, which the position of our teammates also depends on. The choice is made for computational simplicity, especially in the context of the plan generation discussed in Section 4. No fundamental reason would prevent a model from taking our teammates' positions into account and this is a direction for future work.

We will present the model representation and selection algorithm in general form in Sections 5.1 and 5.2 and then discuss their use in robot soccer in Section 5.3.

### 5.1. Model Representation

Opponent models often have the general form of a function from state to actions (perhaps a probability distribution over actions), which are intended to predict the actions the opponents will take (e.g., Carmel and Markovitch, 1998). We will follow this same basic strategy, but predict the resulting state of the opponents rather than the actions taken to get there. We also consider predicting the behavior of a team of agents, rather than a single agent.

Let $p$ be the number of players on a team. For notational simplicity, we will assume our team and the opponent team have the same number of players, but this is not essential to our formulation. We assume that states of the world can be decomposed into three components:

$\mathcal{S}_T^p$ This sequence of $p$ elements represents the state of each of our team members. In other words, each agent's state can be represented as an element of $\mathcal{S}_T$.

$\mathcal{S}_O^p$ The states of the $p$ members of the opponent team.

$\mathcal{S}_W$ The states of the world not represented by $\mathcal{S}_T^P$ or $\mathcal{S}_O^p$.

This state decomposition into world and agents' states is similar to that used by Xuan et al. (2001) to model communication between agents. The decomposition will be used to identify exactly what part of the total state our agent models will take as input and what part of the total state the models will predict. For example, if you had a world described by a sequence of state variables, the decomposition above would correspond to assigning each of the state variables to a particular agent or the world in general. In many such state variable descriptions, this would be a very easy and natural assignment.

Let $\mathcal{R}_O$ represent the set of probability distributions over $\mathcal{S}_O$ and let $\mathcal{A}$ represents the set of sequences of possible actions (including durations, if applicable) our team can take. Conditional plans, where the next action depends on the observed state, are not considered here.

An opponent model is then a function that probabilistically predicts the opponents' future states based on the world state, opponent states, and planned actions of our team. In other words, we define a model $M$ as a function:

$$M\colon \mathcal{S}_W \times \mathcal{S}_O^p \times \mathcal{A} \to \mathcal{R}_O^p \tag{1}$$

Note that our team's state $\mathcal{S}_T^p$ is not part of the inputs to the model. However, the team's state is constrained by the team's actions (an element of $\mathcal{A}$), so some information about the team state is implicitly available to the model. The choice to remove the explicit dependence on $\mathcal{S}_T^p$ is made for computational simplicity. On the other hand, each opponent player's final state distribution may depend on the starting states of *all* the opponent players.

An opponent model of this form can be used to calculate the probability of an opponent ending in a particular state. In particular, given a world state $w \in \mathcal{S}_W$, opponent states $s_i \in \mathcal{S}_O\,(\forall i \in [1,p])$, and a planned team action $a \in \mathcal{A}$, an opponent model $M$ says that the probability for player $j$ being in ending state $e_j \in \mathcal{S}_O$ is

$$P_j[e_j|w, s_1, \ldots, s_p, a, M] := M(w, s_1, \ldots, s_p, a)[j](e_j) \tag{2}$$

We use $P_j$ to represent the probability over ending opponent states for opponent $j$. Notationally, we also consider probability distributions to be functions from the input set to the real numbers.

In contrast to opponent models for game tree search (e.g., Carmel and Markovitch, 1996), our opponent models are predicting not just one action response of an opponent, but the result of a series of interleaved team and opponent actions. Our model is explicitly operating on abstract temporal and action levels, making our model more applicable to environments with continuous or many discrete action opportunities.

### 5.2. Model Selection

Given the description of the opponent models, we can now describe the algorithm for selecting the best matching model. Given the assumption 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 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. When 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 $\{M_1, \ldots, M_m\}$ and then observe. An observation is a tuple of starting

world state $w \in \mathcal{S}_W$, starting states for all the opponent players $s \in \mathcal{S}_O^p$, a planned team action $a \in \mathcal{A}$, and ending states for all opponent players $e := \langle e_1, \ldots, e_p \rangle \in \mathcal{S}_O^p$. 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 with an observation $o = \langle w, s, a, e \rangle$. We want $P[M_i|o]$ for each model $M_i$. Using Bayes' rule we get

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

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 $x$'s ending state based on everyone's starting states. 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 probabilities of a particular set of starting states and planned action are independent of the opponent model. This assumption is questionable since the planned agent actions may depend on the opponent model determined to be the most likely. However, results in Section 6.2 demonstrate we still are able to recognize models correctly.

$$\begin{aligned}
P[M_i|o] &= \frac{P[w,s,a,e|M_i]}{P[o]} P[M_i] \quad \text{(from eq. (3))} \\
&= \frac{P[e|w,s,a,M_i]P[w,s,a|M_i]}{P[o]} P[M_i] \\
&= P[e|w,s,a,M_i] \frac{P[w,s,a]}{P[o]} P[M_i] \quad \text{(assump. 2)} \\
&= \underbrace{P[e_1|w,s,a,M_i]P[e_2|w,s,a,M_i] \ldots P[e_p|w,s,a,M_i]}_{\text{what opponent model calculates (eq. (2))}} \\
&\quad \underbrace{\frac{P[w,s,a]}{P[o]}}_{\text{norm. constant}} \underbrace{P[M_i]}_{\text{prior}} \quad \text{(assump. 1)}
\end{aligned} \tag{4}$$

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.

The preceding computation began with the assumption that the opponent has chosen one of our models and was generating observations from it. Of course, this is only an approximation. However, if one model generally makes better predictions than the others, then that model will be the most likely. Assuming one of the given models is correct is the same type of assumption that is made in a number of statistical machine learning problems. The question we are trying to answer is "Which of the models from this set best explains opponent behavior?" This parallels the question in most machine learning tasks with generative models of "Which hypothesis from this set best predicts the data?"

In addition to the update above, we use weight sharing at the end of each update cycle. A small probability mass (0.1) is added to the probability value for every model and then the distribution is renormalized. This means that if there are $m$ models, a probability $p$ becomes:

$$\frac{p + 0.1}{1 + 0.1m} \tag{5}$$

Weight sharing prevents any model's probability from going arbitrarily close to 0, while not changing which model is most likely on any one update. Weight sharing allows the update process to more quickly capture changes in the opponents behavior (if their behavior switches from one model to another). For example, if the prior and an observation tell us that one model has probability of 1, we still put a probability mass of $\frac{1}{10+m}$ on every other model, meaning that we still believe there is a chance that the opponent team will, in the future, act as described by the model.

Weight sharing also means that more recent observations are weighted more heavily. Each step of weight sharing smooths out the probability distribution. The perturbation caused by an observation (i.e. making one or more models more likely based on what was observed) is smoothed out by the weight sharing steps of other observation updates. More recent observations have gone through less smoothing operations and can therefore have an effect of larger magnitude.

If the opponent is changing or adapting, the coach may be able to track the changes, depending on the speed of adaptation. Of course, a team that knew exactly the algorithm we were using could still conceivably adapt just faster than the coach could keep up with. Applying regret-minimization techniques such as Auer et al. (2002) is an interesting future direction, but as far as the authors know, the complexity of the environment prevents the direct application of any known techniques.

Conceptually, we want an opponent model to represent how an opponent plays defense during setplays. We conjecture that a wide range of decision making systems of the opponent can be roughly captured by a small set of models, but we have not empirically verified this conjecture.

Remember that $p$ is the number of players on a team. Let $\mathcal{L}$ be the set of positions on the field, discretized to 1m. The player state sets $\mathcal{S}_T$ and $\mathcal{S}_O$ are both equal to $\mathcal{L}$, representing to location of a player. The world state $\mathcal{S}_W$ will also be equal $\mathcal{L}$, representing the location of the ball.

The planned ball movement will be the planned actions of our agents. We represent the ball movement as a sequence of locations on the field (an element of $\mathcal{L}^*$). The expected time for each ball movement, with bounds for normal execution, can be calculated based on the empirically determined environment and agent execution parameters, such as time to kick the ball and speed of the ball when passed.

An opponent model is then trying to predict the future location of each opponent will given the ball's current location ($w \in \mathcal{L}$), the opponents' initial positions $s \in \mathcal{L}^p$, and a future path of the ball $a \in \mathcal{L}^*$. In other words, the model answers a question like "Given the positions of the opponents and the ball, if the ball moves like this over the next 2 seconds, where will the opponents be at the end of those 2 seconds?" Formally, we have:

$$M: \underbrace{\mathcal{L}}_{\substack{\text{ball}\\ \text{position}}} \times \underbrace{\mathcal{L}_O^p}_{\substack{\text{opponent}\\ \text{starting}\\ \text{positions}}} \times \underbrace{\mathcal{L}^*}_{\substack{\text{planned}\\ \text{ball}\\ \text{movement}}} \rightarrow \underbrace{(\text{Prob. Dist. over } \mathcal{L})^p}_{\substack{\text{predicted}\\ \text{opponent}\\ \text{positions}}} \quad (6)$$

Equation (6) is an instantiation of Equation (1) for robot soccer. An example application of a soccer opponent model is shown in Figure 9. Here, the model predicts that both opponents move towards the final location of the ball.

Thus far, we have described the format of the opponent model, i.e. *what* must be computed, but not *how* this computation is done. For the implemented system, all player distributions are represented as Gaussians. The models are simply functions which manipulate these distributions in the appropriate way. However, note that the selection algorithm described in Section 5.2 does not depend on this representation.

As an example, one of the models we use has all opponent players moving towards the ball. The function that represents the model adjusts the input distributions by moving the means towards the ball and

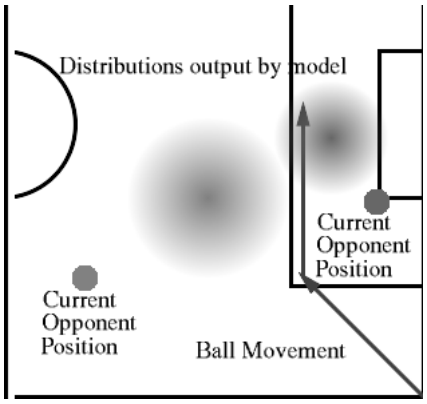


*Figure 9.* An example application of an opponent model. The fuzzy areas represent probability distributions for the two ending locations of the opponent players (shown as dark circles) as the ball moves along a path indicated by the arrows.

increasing all the variances. Section 6.1 discusses the set of models we use for the empirical validation in more detail.

In addition, a decision must be made about how to generate observations from the stream of data being received about the opponent positions. Clearly, if an observation tuple is generated every cycle we will be violating the independence assumption of the naive Bayes update, as well as giving the models little information (in terms of the ball movement) with which to work. On the other hand, the more observations the coach gets, the easier it is to correctly identify the correct model. To balance these competing factors, we decided to create an observation every time the agent who is controlling the ball changes. An agent is considered to be controlling the ball if (i) the agent is the closest player to the ball and (ii) the agent can kick the ball. A given observation can cover anywhere from approximately 5 to 50 cycles (one half to five seconds) of movement.

## 6. Empirical Results

This section presents experiments exploring the effectiveness of the planning approach and the use of opponent models.

### 6.1. General Setup

For all empirical experiments, we first needed to create a set of opponent models. In all of the models, the distribution of each player's 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 created five models for the empirical evaluation. This set of models represent fairly distinct styles of movements by the opponents. 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 0.6m closer to the ball.[6] 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.

These models are *not* for the opponent's behavior throughout an entire simulated soccer game. The models are only intended to capture the way the opponents move for the 5–20 seconds in which our team executes a set play from a dead ball situation. The models do not capture any actions that the opponents take with the ball, or how they play defense more generally.

### 6.2. Model Recognition

This experiment looks at the model recognition algorithm. The models are abstractions over player movements. We want to verify that the recognition algorithm over these models can be used to correctly recognize a team that acts approximately like a given model. We use several assumptions during the probability updates, and if those assumptions are extensively violated, the recognition algorithm will fail to work as predicted. Further, we want to explore how long it takes for the naive Bayes based recognition algorithm to identify the correct model. The coach is the only agent doing the recognition since it has the global view of the field.

[6] The players max speed is 1m/cycle.

Before looking at how well each model can be recognized out of this set of models, we must first understand how well any recognition algorithm could expect to do. We call this the *separability* of a set of models. If two models make similar predictions in most cases, it will be difficult to recognize one model over the other and we should not expect any recognition algorithm to get near perfect accuracy. The concept of separability will give us a standard to compare how well the models are being recognized in the real system. While separability seems to be a basic statistical concept, the author is not aware of standard definitions or calculations which fit this problem.

Separability will of course be a function of an *entire set* of models, not a property of any one model. Also, separability must be a function of the number of observations; as we get more information, we expect to be able to identify the correct model more often.

For illustration, consider a simple example where you have two sets of models of coins. In the first set, one model says that heads comes up 99% of the time and the other says heads comes up 1%. For the second set, the models say 51% and 49%. Separability asks the question: if the world is exactly described by one of the models in our set (but we don't know which one), how does the number of observations affect the probability we will identify the correct model? Clearly, we are much more likely to identify the correct model for the first set than the second set because the predictions are so different.

We will develop the concept of separability in four stages. First we will consider the separability of two distributions given one observation, then the separability of two distributions given multiple observations, then the separability of a set of distributions, and finally the separability of a set of models.

Start with two distributions $A$ and $B$ over two dimensions. The question we are interested in is: if we are seeing an observation probabilistically generated from $A$, what is the probability that the naive Bayes update (starting with a uniform prior) will return with $A$ being the most likely distribution? Equivalently, what is the probability mass of $A$ in the area where the probability distribution function (i.e. pdf) of $A$ is greater than the pdf of $B$? Of course, we are also interested in the case where $B$ is the distribution generating observations, and in general these probabilities can be different.

The concept of separability we are interested in here is similar to relative entropy or Kullback-Leibler distance (Cover and Thomas, 1991). The relative entropy of distribution $A$ to distribution $B$ is (where $f_A(x)$

is the pdf of $A$ at $x$):

$$D(A||B) := \int f_A(x) \log \frac{f_A(x)}{f_B(x)} dx \tag{7}$$

The important difference is that Kullback-Leibler distance is considering the ratio of $f_A$ to $f_B$. In our update, we only care whether $f_A$ or $f_B$ is larger; that is, if the wrong distribution comes out as more likely in our update, we don't care how wrong it is. In other words, our loss function is binary (correct or incorrect) and Kullback-Leibler is targeted for a loss function which is not. More precisely, if $I_{A>B}(x)$ is an indicator function for whether $f_A(x) > f_B(X)$, then our separability of $A$ from $B$ is

$$\int f_A(x) I_{A>B}(x) dx \tag{8}$$

If one considers the models in the context of how they are used, then the appropriate loss function may not be binary as indicated here. The loss function should represent how bad it is to use one model when another one is the true model. As will be shown below, computing the separability of *models* is not trivial, nor is computing the true loss function given the complicated use of the models in Section 4.1. Therefore, we simplify the separability computation by assuming a binary loss function.

Now consider seeing multiple observations instead of a single one. Once again, we are interested in the probability that $A$ will have the highest posterior after the naive Bayes update on all of the observations. This probability is challenging to solve for analytically, but Monte Carlo simulation can estimate it.

Our concept of separability extends naturally to a set of distributions rather than just two distributions. We still want to measure the chance that the correct distribution has the highest probability. We can again use Monte Carlo simulation to estimate the probability that the correct distribution will be recognized for any given number of observations.

Finally, the opponent models are not simple distributions. The models are functions from starting world states, starting states of the opponents, and a planned team action to distributions of the opponents' states. We use an empirical test to estimate separability of *models*. For each observation $o_1, \ldots o_k$ from real game play, we repeatedly generate a series of artificial observations

$$\langle w, s, a, e^1 \rangle \ldots \langle w, s, a, e^n \rangle \tag{9}$$

where

- $w \in \mathcal{S}_W$ is the starting world state .

- $s \in \mathcal{S}_O^p$ is the set of opponents' starting states.
- $a \in \mathcal{A}$ is the planned team action.
- $e^i \in \mathcal{S}_O^p$ is a set of opponents' ending states.
- $n$ is the number of observations for which we want to estimate the separability.

$w$, $s$, and $a$ are taken from the real observation $o_i$. Each set of ending opponents' states $e^1 \dots e^n$ is sampled from the distributions output by the correct model (the model for which we want to estimate the probability of the naive Bayes being correct). For each sequence of artificial observations, the update is performed. Averaging over all series of observations, we can estimate the probability of a model being correctly recognized, given $n$ observations.

Figure 10 shows, for each model, the probability of that model being correctly recognized as a function of the number of observations (i.e. separability). One can see that if the models perfectly capture the opponents, after only a few updates, the recognition accuracy should be quite high (more than 85%).

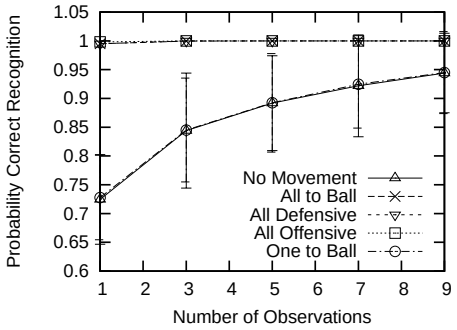


*Figure 10.* Separability of the models. The "No Movement" and "One to Ball" lines are the lower two lines. Error bars of one standard deviation are shown on only those two lines because the error bars are too small to be visible on the others.

Given the accuracy we would expect if our models were perfect, we can empirically see how well our recognition algorithm actually does. For each model, we programmed teams of agents to act as close as we could manage to the behavior predicted by the model. Note that we can not make this behavior perfect because of the partial observability and dynamics of the world. We then ran our team and coach against each of these programmed teams and recorded all the observations obtained. For each number of observations $n$, we examine all contiguous sequences of $n$ observations. For each sequence of observations, we perform the

naive Bayes update (starting from a uniform prior). For each model, the empirical recognition accuracy is the percentage of the time that the correct model came up as most likely after that series of observations.

Figure 11 summarizes the recognition accuracy of the models. For most models we achieve 90% recognition after just 4 observations. The accuracies track the separability from Figure 10 with some exceptions. First, there is more confusion among all models for the lowest couple of observations. This suggests that the tails of the distributions are heavier than what the Gaussian model suggests, or possibly that the distributions are multi-modal. Second, the One to Ball model is confused with the other models less than the separability indicates, which probably means that the actual variation for the team is less than that suggested by the model. Finally, the All Offensive model never achieves the near 100% performance suggested by the separability. This result reveals something missing from the model. Namely, during the actual executions, the players on the team will not position themselves offsides. The details of the offsides rule in soccer are not important, but basically it prevents the players from moving too far to the offensive side. The offsides rule, which is ignored by the model, prevents the model's predictions from being completely accurate. Overall, note that the recognition accuracy is quite high after very few observations.

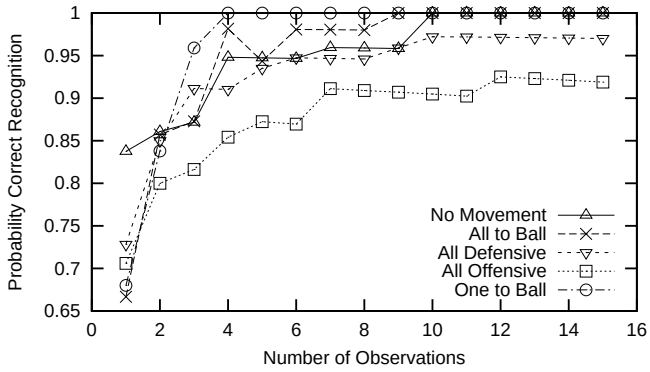


*Figure 11.* Given a number of observations and a model that the opponent is approximately executing, this graph shows the percentage of time a contiguous sequence of real observations results in the correct model being most likely after our naive Bayes update. This graph can be compared to the separability from Figure 10, but it should be noted that the axes have different scales.

The exact separability and rate that the accuracy increases will of course change given the set of models that one chooses. However, the observed accuracies tracked the theoretical separability quite well, suggesting that the assumptions made in the model recognition algorithm were good enough. Further, the recognition accuracies increased quite

quickly, which is at least suggestive that other model sets may have similarly quick recognition.

### 6.3. Models' Effect on Planning

Given that we have opponent models that can be selected based upon observations, we now need to examine the effect of those models on the output of the planning process. If the opponent models are meaningful, then the resulting plans should be different for the different models. It is somewhat tricky to isolate just this one effect from the rest of the system. In order to evaluate the differences in the plans produced using opponent models, we compare paths by looking at the area between the paths. For example, in Figure 12, 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, and consequently, how different two plans are. Therefore, the median area difference[7] between a set of plans expresses roughly how much variation there is in a set of plans. The exact numbers are not especially meaningful, but are useful as comparisons of different sets of plans.

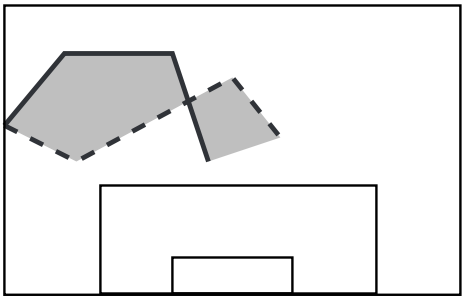

*Figure 12.* Example of the area between paths. The two paths are the solid line and the dotted line and the shaded area is the area between them.

First we look at the variation in our planning seeds. The planning seeds are designed to be far apart in the space of possible plans, so the variation gives some idea of the maximum range we could expect the plans generated by the system to vary. As shown in Table III, the median area difference is 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. The difference between opponent models is somewhat lower than the median difference between the planning seeds. This lower difference is not surprising because the evaluation of a path depends strongly upon

[7] We use median rather than mean because the distributions of path areas have heavy tails.

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 should be higher.

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

| Plan Set | Median Area Difference | # Comparisons |
|---|---|---|
| Planning seeds | 315 | 138 |
| Across opponent models | 138 | 223 |
| For one opponent model | 0 | 6625 |
| For one opponent model (unique plans) | 116 | 3073 |

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 the planner on a set of 25 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[8] returned over these 53 trials, the median area of 116 is still smaller than the median area between plans returned by different models. This result suggests that, as expected, the model used causes more variation in the output of the planner than the random variation in the hillclimbing.

### 6.4. Total Game Effect

A team's overall performance in a simulated soccer game is a product of many tightly interacting factors. The experiments in this section are designed to show that adaptive setplays can have a positive impact on the overall team performance. It is not a thorough evaluation of *when* and *why* the adaptive setplays have an impact. In other words, these experiments are an existence proof that the representation and algorithms that we are proposing can have a positive effect on the performance of the team.

Over the course of a simulation game, setplays are executed for a fairly small percentage of the total time. Therefore, the absolute effect of setplays on the final score difference is expected to be small even if the setplays are significantly better than what was present before.

[8] At most 2 different plans were returned for a given model and problem.

Any significant effect of the overall score of a team by improving the setplays is an achievement.

The plan execution algorithm was fully implemented in the ATT-CMUnited2000 simulation soccer team (Riley et al., 2001). The team ATT-CMUnited2000 was based on CMUnited99 (Stone et al., 2000) which has fixed setplay plans for each type of setplay (goal kick, kick in, free kick, etc.). We ran ATT-CMUnited2000 using the old fixed setplays and the new adaptive setplays, playing against CMUnited99 in both cases. For the new adaptive setplays, the opponent model set described in Section 6.1 was used. The results are shown in Table IV.

Table IV. Comparison of fixed setplays from CMUnited99 to adaptive setplays using MASTN. With a standard one tailed $t$-test, the difference in goals scored is not significant, while the difference in goals against is ($p < .01$). All games were against CMUnited99.

| | # Games | Mean goals scored | Mean goals against |
|---|---|---|---|
| CMUnited99 fixed setplays | 33 | 2.45 | 0.30 |
| Planned, MASTN based setplays | 56 | 2.55 | 0.18 |

The results show that the new setplays have a small but significant effect on the overall performance of the team. The effects on goals against seems to occur for two reasons. First, setplays such as goal kicks and goalie catches can be dangerous times for the team because the ball starts so close to the goal. Executing a good play to get the ball upfield can get out of these dangerous situations. Second, a good offense can be the best defense. If the team spends more time attacking, the opponent has less opportunities to score goals. Overall, this result is good given the small proportion of the time of the game occupied by setplays.

We wanted to further test the effectiveness of the adaptive setplays. However, there is considerable effort in linking the plan execution algorithm to the behavior architecture of a player. The standard coaching language CLang (Chen et al., 2001) was created around the time we were finishing the previous experiments. We therefore created an algorithm to convert the MASTN into CLang condition-action rules. This conversion allows the setplays to be used with any team that understands CLang rather than being restricted to ATT-CMUnited2000.

Translating the MASTN plan into CLang requires that the effects of actions are encoded into conditions of rules. Since CLang supports rule conditions dealing with ball and player positions, most action effects in

MASTNs can be encoded effectively into CLang conditions. However, some aspects of the execution algorithm can not be encoded because of limitations in CLang. For example, since CLang has no action representing where an agent should be looking, the agent can not be told to look to where a relevant action should be taking place. Further, the temporal expressibility is CLang is such that the propagation of temporal constraints can not be done, though ordering of events can still be maintained.

The coachable team ChaMeleons (Carpenter et al., 2002), which was created at Carnegie Mellon, was used as the recipient of our planning advice. The opponent used was Gemini from the Tokyo Institute of Technology (Coradeschi and Tadokoro, 2002). We used Gemini for two reasons. First, Gemini was the opponent in the RoboCup 2001 coach competition, allowing us to compare performance to what was observed there. Second, since Gemini was not created by us, nor do we have any knowledge of its behavior algorithms, this provides a more independent test of the setplays.

The coach needs to know the formation, or arrangement of players on the field, in order to assign players to roles. Therefore, our coach also sends the players a formation, i.e. a spatial assignment. Details about the structure of the formation and how it is learned can be found in Riley et al. (2002).

We ran a series of simulation games under different conditions.[9] In all cases, the set of opponent models described above was used. Each experimental condition was run for 30 games and the average score difference (as our score minus their score) is reported. Therefore a negative score difference represents losing the game and a positive score difference is winning. All significance values reported are for a two tailed $t$-test.

Three sets of games were run: a baseline without the coach, the coach just sending a formation, and the coach sending a formation and planning setplays. Since there are many interacting factors affecting the performance of a simulated robot soccer team, we are more interested in the improvement that the setplays has on the coached team rather than the absolute win/loss value of the coached team against the opponent. Table V shows the results.

The use of the setplays significantly ($p < 0.01$) improves the performance of the team, both over just the use of the formation and over

[9] In all of these experiments, we slowed the server down to 3-6 times normal speed so that all agents could run on one machine. This was done for convenience for running the experiments. We tried to verify that agents were not missing cycles and while this setup shouldn't affect outcomes compared to running on several machines, the design of the server makes it impossible to say for sure.

Table V. Mean score difference under various experimental conditions. The score difference reported is the coached team score minus opponent score. The interval next to the score is the 95% confidence interval.

| Condition | Score Difference |
|---|---|
| Baseline (without setplays) | -6.5 [-7.2, -5.9] |
| With formation | -9.1 [-10.0, -8.2] |
| With setplays and formation | -4.2 [-4.9, -3.5] |

not using either a formation or setplays. The effect here is larger than in the previous experiment which compared fixed vs. adaptive setplays in ATT-CMUnited2000 since we are comparing no set plays to our adaptive setplays in this case.

## 7. Related Work

MASTNs refine Simple Temporal Networks (STNs) to be used in the multi-agent case. STNs have been used to solve scheduling problems (e.g., Morris and Muscettola, 2000) and the execution algorithm we present uses the algorithms presented by Muscettola et al. (1998) as subroutines.

Multi-agent plan representations have been suggested by a number of other researchers. Several models of multi-agent systems have been proposed (Boutilier, 1999, Peshkin et al., 2000, Bernstein et al., 2000, Xuan et al., 2001, Pynadath and Tambe, 2002). For all of these models, solving the model yields a joint action policy for the agents. This policy can be seen as a universal plan for the agents. However, the dimensionality of the model means that such polices quickly become difficult to construct or communicate. Therefore, a naive use of the models would not be an effective way to create and communicate about team plans. Indeed, one of the motivations for creating the COM-MTDP model (Pynadath and Tambe, 2002) was to be able to evaluate algorithms which use more efficient reasoning processes. The MASTNs that we introduce are one compact way to represent a joint plan.

Bowling et al. (2004) introduce tactics, plays, and play books as multi-agent plans. In the context of small size robot soccer, they define tactics as single agent, primarily reactive behaviors. Plays are specifications of roles by the sequence of tactics they should be performing. The play book is a collection of plays. This approach addresses a number

of orthogonal issues to those addressed by the MASTNs. Most importantly, the strength of MASTNs is in handling agents whose beliefs about the world may be inconsistent and allowing them to coordinate successfully. While the play-based approach must consider multi-agent problems such as role assignment, one central controller makes those decisions.

Teamwork theories must also consider representing and manipulating multi-agent plans. SharedPlans (Grosz and Kraus, 1996) and Joint Intentions (as expressed in GRATE* (Jennings, 1995)) are normative specifications of mental attitudes of agents in a team. These teamwork theories are meant to guide the design of agents. Both theories use the concept of a recipe where agents are committed to performing possibly ordered actions. While both approaches catalog recipe failures and specify how to deal with them, MASTNs provide a more specific framework for representing and reasoning about coordination and failure through temporal constraints. In particular, those approaches specify how the beliefs, desires, and intentions should change in response to team events, but the approaches leave open how those beliefs, desires, and intentions are translated into actions. Therefore, while the SharedPlans and Joint Intentions frameworks are more general, MASTNs provide more of a solution if one can represent the needed coordination and failure modes in the temporal constraints of the network.

STEAM (Tambe, 1997) draws on both SharedPlans and Joint Intentions. The key innovation is the introduction of team operators. Each agent maintains its own perception of the state of execution of these team operators. STEAM is then a system for maintaining as much consistency as needed and possible among these operators. Thus, our MASTN representation and algorithms can be seen as an application of these same concepts into a representation with temporal constraints.

Intille and Bobick (1999) also use temporal constraints to express coordination. However, their temporal constraints are fuzzy and qualitative, such as "A around B" meaning that event A should occur around the same time as B. The other major difference is that the nodes in their temporal network represent agent goals, not particular events. They apply their representation to plan recognition in records of human American football. In other words, similar temporal constraints and reasoning are used to solve a different problem.

Doyle et al. (1986) have examined inserting perceptual expectations into plans based on preconditions and post-conditions. These are similar to the node precondition monitoring that we do during MASTN execution.

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 MASTN representation and execution algorithm.

Agent modeling has been an important topic in computer science and the literature available is correspondingly vast. A number of authors have explored the use of opponent models in playing two person, finite, perfect information games (e.g., Iida et al., Carmel and Markovitch, Donkers, 1993, 1996, 2003). During the search of the game tree, the models predict (perhaps probabilistically) what action the opponent will take and the search attempts to find a best response. This article addresses a significantly different environment, specifically a multi-agent, continuous state and action, and partial information game.

Using an agent's own behavior representation to infer opponent actions has been used by Tambe and Rosenbloom (1995) in an air combat domain and by Laird (2001) in Quake, a real-time first person shooter computer game. We explicitly avoid having our agent models be based on the agent's internal execution policy because the coach is not an agent that executes actions in the environment and we want to provide as much generality as possible in our models.

Section 5 argues that selecting between opponent models online rather than trying to learn from scratch is a promising avenue for dealing with sparsity of data in an online setting. This same idea is discussed and tested in a predator-prey domain by Denzinger and Hamdan (2004). Their models, which they call "stereotypes," have a very different structure and they use different selection algorithms, but end goal is the same.

Work on agent modeling has also occurred in the simulated robotic soccer community. Wünstel et al. (2001) use self organizing maps to classify the movements of agents. Miene et al. (2004) use coarsely discretized state and action descriptions to arrive at a *qualitative* motion model which can be used to predict impending offside situations. Also in RoboCup, but for the small size robots, Han and Veloso (2000) use Hidden Markov Models (HMM) to recognize behaviors of robots, with each HMM representing a model of a behavior of a robot. These models are focused primarily on recognizing the behavior of an agent or team and not on how to use such a model to improve performance.

Other groups have used coaches to improve performance in the robot soccer simulator. One of the first teams to use an online coach was Kasugabito (Takahashi, 2000). Based on the score difference, time remaining, and ball's path, the online coach would adjust the team's formation.

Since then, formation learning and switching has been a popular approach. In addition to our previous work (Riley et al., 2002), several other teams have tried various approaches to learning and using formations (Habibi et al., 2002, Drücker et al., 2001). While there are differences among these representations and algorithms, all have the same goals and perform reasonably well.

One of the first systems to analyze a team's play to provide advice was ISAAC (Raines et al., 2000). ISAAC used decision tree learning to identify rules describing the conditions for when goals were and were not scored for and against a team. ISAAC had no automated way for the agents to incorporate advice; the output was descriptive natural language which human developers could use to change their team.

Online coaching with automatically incorporated advice gained importance in RoboCup 2001 with the creation of the online coach competition. In addition to our work (Riley, 2005), the Dirty Dozen team published opponent modeling research (Steffens, 2002). The focus of the work was Feature-Based Declarative Opponent-Modeling (FBDOM) where opponent models use features that are associated with actions. These models are created by hand and can then be matched to observed behavior similar or used to imitate a team.

In addition to the formation learning and adaptation techniques mentioned above, we know of two other coaches that make use of machine learning. The UTAustinVilla coach (Kuhlmann et al., 2005) learns and uses similar opponent models to ones described in Riley (2005), though with some important representational differences. The Sharif Arvand coach (Ahmadi et al., 2003) uses a two-layered case based reasoning approach to predict the future movements of the player and the ball, though it is not specified in that article how predictions are translated into useful advice.

## 8. Conclusion

We have presented an approach to planning by a coach agent which is adaptive to the current adversary. The main technical contributions of this article are:

1. Multi-Agent Simple Temporal Networks as a multi-agent plan representation and an associated distributed execution algorithm. The plan representation expresses temporal coordination and monitoring in a distributed fashion.

2. An algorithm for generating a multi-agent plan for agent movements in the MASTN plan representation.

3. An opponent model representation which probabilistically predicts movements of agents.

4. A method for adaptation to adversarial strategies based on a naive Bayes classifier over 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 addition to the empirical results presented here, in the games at the RoboCup competitions, it was evident that our team benefited from adaptive setplays. Our system created a variety of setplay plans in adaptation to completely unknown opponent teams.

We implemented our coaching approach in simulated robot soccer instead of a simple toy domain, which leads us to infer the applicability to other domains with the following characteristics:

- The MASTN representation and execution algorithm make several assumptions about the environment:
  - Coordination constraints can be expressed in temporal constraints.
  - Each agent can get some information about the execution of all events in the plan, though it is explicitly allowed that the information across agents is inconsistent. This information could come from observation or communication.
  - All parallelism in the plan is across agents; each agent executes one macro-action (i.e. one plan step) at a time.
- The plan generation algorithm requires the user to define an evaluation function over plans which is suitable for hillclimbing. Further, temporal bounds on the normal execution of the steps of the plan are required to create the temporal constraints in the MASTN.
- The opponent model representation and recognition algorithm assume that the state of the world can be decomposed into states for our agents, the opponents, and the rest of the world. Further, the models were created for spatial states and are therefore probably most appropriate in environments with a strong spatial component.

This work opens several interesting future research directions. Currently, during execution, the agents follow a single plan. Storing alternate plans and intelligently adding monitors for these plans as done by Veloso et al. (1998a) could make agents switch between plans during execution if conditions change.

MASTNs showed to be a successful basis for a multi-agent plan representation. Adding additional information to the representation to allow the agents to reason about expected perceptions of events and needed communication is an interesting direction for extending this work.

While our system learns which model best matches the current opponent, learning the models from which to select is another interesting challenge. The predictions of the models have clear semantics and we have already defined how to evaluate how well the models match observed behavior. These two properties of the models and our selection algorithms should help in the development of model learning algorithms.

This article has discussed our planning and opponent modeling approach and its use in simulated robot soccer. Empirical results show the effectiveness at several levels. This work is a contribution to the general problem of how a coach agent can effectively advise a team of agents to respond and adapt to adversaries.

## Acknowledgements

This research was sponsored in part by the United States Air Force under Grants Nos F30602-98-2-0135 and F30602-00-2-0549 and by an NSF Fellowship. A portion of this work was also done while Patrick Riley was at Draper Labs and the authors especially wish to thank Michael Cleary and Deb Dasgupta for their insight and help during the initial implementation. 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, the NSF, Draper Labs, or the US Government.

## References

Ahmadi, M., A. K. Lamjiri, M. M. Nevisi, J. Habibi, and K. Badie: 2003, 'Using a Two-Layered Case-Based Reasoning for Prediction in Soccer Coach.'. In: *Proceedings of the International Conference on Machine Learning; Models, Technologies and Applications (MLMTA'03)*. pp. 181–185.

Auer, P., N. Cesa-Bianchi, Y. Freund, and R. E. Schapire: 2002, 'The non-stochastic multi-armed bandit problem'. *SIAM Journal on Computing* **32**(1), 48–77.

Bernstein, D. S., S. Zilberstein, and N. Immerman: 2000, 'The Complexity Of Decentralized Control Of Markov Decision Processes'. In: *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2000)*. pp. 32–37.

Boutilier, C.: 1999, 'Sequential Optimality and Coordination in Multiagent Systems'. In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*. pp. 478–485.

Bowling, M., B. Browning, and M. Veloso: 2004, 'Plays as effective multiagent plans enabling opponent-adaptive play selection'. In: *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*. pp. 376–383.

Carmel, D. and S. Markovitch: 1996, 'Incorporating Opponent Models into Adversary Search'. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*. Portland, OR, pp. 120–125.

Carmel, D. and S. Markovitch: 1998, 'Model-based Learning of Interaction Strategies in Multiagent Systems'. *Journal of Experimental and Theoretical Artificial Intelligence* **10**(3), 309–332.

Carpenter, P., P. Riley, G. Kaminka, M. Veloso, I. Thayer, and R. Wang: 2002, 'ChaMeleons-01 Team Description'. In: A. Birk, S. Coradeschi, and S. Tadokoro (eds.): *RoboCup-2001: Robot Soccer World Cup V*, No. 2377 in Lecture Notes in Artificial Intelligence. Berlin: Springer-Verlag, pp. 503–506.

Chen, M., K. Dorer, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostiadis, J. Kummeneje, J. Murray, I. Noda, O. Obst, P. Riley, T. Steffens, Y. Wang, and X. Yin: 2001, 'Soccer Server Manual'. RoboCup Federation, `http://sserver.sourceforge.net/`.

Coradeschi, A. B. S. and S. Tadokoro (eds.): 2002, *RoboCup-2001: Robot Soccer World Cup V*. Berlin: Springer Verlag.

Cover, T. and J. Thomas: 1991, *Elements of Information Theory*, Wiley Series in Telecommunications. New York: John Wiley & Sons.

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

Denzinger, J. and J. Hamdan: 2004, 'Improving Modeling of other Agents using Stereotypes and Compactification of Observations'. In: *Proceedings of the Third Autonomous Agents and Multi-Agent Systems Conference*. pp. 1414–1415.

Donkers, H. H. L. M.: 2003, 'Nosce Hostem - Searching with Opponent Models'. Ph.D. thesis, Universiteit Maastricht. ISBN 90-5278-390-X.

Doyle, R., D. Atkinson, and R. Doshi: 1986, 'Generating perception requests and expectations to verify the executions of plans'. In: *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*. pp. 81–88.

Drücker, C., S. Hübner, E. Schmidt, U. Visser, and H.-G. Weland: 2001, 'Virtual Werder'. In: P. Stone, T. Balch, and G. Kraetzschmar (eds.): *RoboCup 2000: Robot Soccer. World Cup IV*, Vol. 2019 of *Lecture Notes in Artificial Intelligence*. pp. 421–424, Springer-Verlag.

Gmytrasiewicz, P. J. and E. H. Durfee: 1995, 'A Rigorous, Operational Formalization of Recursive Modeling'. In: *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*. pp. 125–132.

Grosz, B. and S. Kraus: 1996, 'Collaborative Plans for Complex Group Action'. *Artificial Intelligence* **86**(2), 269–357.

Habibi, J., E. Chiniforooshan, A. HeydarNoori, M. Mirzazadeh, M. Safari, and H. Younesi: 2002, 'Coaching a Soccer Simulation Team in RoboCup Environment'. In: *Proceedings of the First EurAsian Conference on Information and Communication Technology*. pp. 117–126, Springer-Verlag.

Han, K. and M. Veloso: 2000, 'Automated Robot Behavior Recognition Applied to Robotic Soccer'. In: J. Hollerbach and D. Koditschek (eds.): *Robotics Research: the Ninth International Symposium*. London: Springer-Verlag, pp. 199–204.

Iida, H., J. W. H. M. Uiterwijk, and H. J. van der Herik: 1993, 'Opponent-Model Search'. Technical Report CS 93-03, Universiteit Maastricht.

Intille, S. and A. Bobick: 1999, 'A framework for recognizing multi-agent action from visual evidence'. In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*. pp. 518–525, AAAI Press.

Jennings, N. R.: 1995, 'Controlling Cooperation problem solving in industrial multi-agent systems using joint intentions'. *Artificial Intelligence* **75**(2), 195–240.

Kitano, H., M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada: 1997, 'The RoboCup synthetic agent challenge'. In: *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*. San Francisco, CA, pp. 24–49.

Kuhlmann, G., P. Stone, and J. Lallinger: 2005, 'The Champion UT Austin Villa 2003 Simulator Online Coach Team'. In: D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor (eds.): *RoboCup-2004: Robot Soccer World Cup VIII*, Vol. 3276. Berlin: Springer Verlag, pp. 636–644.

Laird, J. E.: 2001, 'It Knows What You're Going To Do: Adding Anticipation to a Quakebot'. In: *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001)*. pp. 385–392.

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

McAllester, D. and P. Stone: 2001, 'Keeping the Ball from CMUnited-99'. In: P. Stone, T. Balch, and G. Kraetszchmar (eds.): *RoboCup-2000: Robot Soccer World Cup IV*. Berlin, pp. 333–338, Springer Verlag.

Miene, A., U. Visser, and O. Herzog: 2004, 'Recognition and Prediction of Motion Situations Based on a Qualitative Motion Description'. In: D. Polani, A. Bonarini, B. Browning, and K. Yoshida (eds.): *RoboCup 2003: Robot Soccer World Cup VII*, Vol. 3020 of *Lecture Notes in Artificial Intelligence*. pp. 77–88, Springer-Verlag.

Morris, P. and N. Muscettola: 2000, 'Execution of Temporal Plans with Uncertainty'. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*. pp. 491–496, AAAI Press/The MIT Press.

Muscettola, N., P. Morris, and I. Tsamardinos: 1998, 'Reformulating Temporal Plans for efficient execution'. In: *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*. pp. 444–452.

Noda, I., H. Matsubara, K. Hiraki, and I. Frank: 1998, 'Soccer server: A tool for research on multiagent systems'. *Applied Artificial Intelligence* **12**(2–3), 233–250.

Peshkin, L., K.-E. Kim, N. Meuleau, and L. P. Kaelbling: 2000, 'Learning to Cooperate via Policy Search'. In: *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2000)*. pp. 489–496.

Pynadath, D. and M. Tambe: 2002, 'The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models'. *Journal of Artificial Intelligence Research* **16**, 389–423.

Raines, T., M. Tambe, and S. Marsella: 2000, 'Automated Assistant to Aid Humans in Understanding Team Behaviors'. In: *Proceedings of the Fourth International Conference on Autonomous Agents (Agents-2000)*. pp. 419–426.

Reece, G. and A. Tate: 1994, 'Synthesizing protection monitors from casual structure'. In: *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*. Chicago, IL, pp. 146–151.

Riley, P.: 2005, 'Coaching: Learning and Using Environment and Agent Models for Advice'. Ph.D. thesis, Computer Science Dept., Carnegie Mellon University. CMU-CS-05-100.

Riley, P., P. Stone, D. McAllester, and M. Veloso: 2001, 'ATT-CMUnited-2000: Third Place Finisher in the RoboCup-2000 Simulator League'. In: P. Stone, T. Balch, and G. Kreatzschmarr (eds.): *RoboCup-2000: Robot Soccer World Cup IV*, No. 2019 in Lecture Notes in Artificial Intelligence. Berlin: Springer, pp. 489–492.

Riley, P., M. Veloso, and G. Kaminka: 2002, 'An Empirical Study of Coaching'. In: H. Asama, T. Arai, T. Fukuda, and T. Hasegawa (eds.): *Distributed Autonomous Robotic Systems 5*. Springer-Verlag, pp. 215–224.

Steffens, T.: 2002, 'Feature-Based Declarative Opponent-Modelling in Multi-Agent Systems'. Master's thesis, Institute of Cognitive Science Osnabrück.

Stentz, A.: 1994, 'Optimal and Efficient Path Planning for Partially Known Environments'. In: *Proceedings of IEEE International Conference on Robotics and Automation*. pp. 3310–3317.

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

Stone, P., P. Riley, and M. Veloso: 2000, 'The CMUnited-99 Champion Simulator Team'. In: Veloso, Pagello, and Kitano (eds.): *RoboCup-99: Robot Soccer World Cup III*, No. 1856 in Lecture Notes in Artificial Intelligence. Berlin: Springer, pp. 35–48.

Stone, P. and M. Veloso: 1999, 'Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork'. *Artificial Intelligence* **110**(2), 241–273.

Stone, P., M. Veloso, and P. Riley: 1999, 'The CMUnited-98 Champion Simulator Team'. In: Asada and Kitano (eds.): *RoboCup-98: Robot Soccer World Cup II*, No. 1604 in Lecture Notes in Artificial Intelligence. Springer, pp. 61–75.

Takahashi, T.: 2000, 'Kasugabito III'. In: Veloso, Pagello, and Kitano (eds.): *RoboCup-99: Robot Soccer World Cup III*, No. 1856 in Lecture Notes in Artificial Intelligence. Berlin: Springer-Verlag, pp. 592–595.

Tambe, M.: 1997, 'Towards Flexible Teamwork'. *Journal of Artificial Intelligence Research* **7**, 83–124.

Tambe, M. and P. Rosenbloom: 1995, 'RESC: An approach for dynamic, real-time agent tracking'. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*. pp. 103–111.

Veloso, M., M. Bowling, and P. Stone: 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*, Vol. 3839. Boston.

Veloso, M., M. Pollack, and M. Cox: 1998a, 'Rationale-based monitoring for planning in dynamic environments'. In: *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*. Pittsburgh, PA, pp. 171–180.

Veloso, M., P. Stone, K. Han, and S. Achim: 1998b, 'CMUnited: A Team of Robotic Soccer Agents Collaborating in an Adversarial Environment'. In: H. Kitano (ed.): *RoboCup-97: The First Robot World Cup Soccer Games and Conferences*. Berlin: Springer Verlag, pp. 242–256.

Weiss, G. (ed.): 1999, *Multiagent systems : a modern approach to distributed artificial intelligence*. Cambridge, Mass.: MIT Press.

Wünstel, M., D. Polani, T. Uthmann, and J. Perl: 2001, 'Behavior Classification with Self-Organizing Maps'. In: *RoboCup-2000*, Vol. 2019 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, pp. 108–118.

Xuan, P., V. Lesser, and S. Zilberstein: 2001, 'Communication Decisions in Multi-Agent Markov Decision Processes: Model and Experiments'. In: *Proceedings of*

*the Fifth International Conference on Autonomous Agents (Agents-2001)*. pp. 616–623.