Multi-Agent Teamwork, Adaptive Learning, and Adversarial Planning in Robocup Using a PRS Architecture

Tommaso F. Bersano-Begey, Patrick G. Kenny, and Edmund H. Durfee
The University of Michigan Artificial Intelligence Laboratory
ATL building, Ann Arbor, Michigan 48109
(tombb@engin.umich.edu, pkenny@umich.edu, durfee@umich.edu)
(web: ai.eecs.umich.edu/robocup)

ABSTRACT

Our approach for the Robocup97 competition is to emphasize teamwork among agents by augmenting reactions (based on awareness of the current situation) with predictions (based on predefined multi-agent maneuvers). These predictions are accomplished by allowing agents to cooperatively accomplish predefined plans, which are elaborated reactively and hierarchically to ensure responsiveness to changing circumstances. By supporting the run-time construction of plans, our approach simplifies the introduction of new plans, strategies, and actions, and produces a framework for dynamic adaptation and plan recognition through automatically generating belief networks. Our implementation is built on top of UM-PRS, a procedural reasoning system architecture for real-time environments, which allows specifying, executing, and integrating plans based on subgoaling and preconditions.

1.Introduction

Competing in Robocup97 implies building a multi-agent system which behaves in real-time in a dynamically changing environment. Goals and situations change frequently, and information is often partial or noisy. In the PreRoboCup-96 competition, teams with mostly simple hand-coded behaviors performed better than ones which attempted to use more complex approaches in learning, planning and opponent's modeling, apparently because it is much more important to react quickly to the current situation than to analyze and plan for future situations that might never materialize.

However, as RoboCup competitors are endowed with a full repertoire of reactive behaviors, it is likely that what will set teams apart will be the added capabilities they bring to the table. Our hypothesis that while being aware of and acting appropriately for the current situation is the most critical capability for an agent, an agent that is able to behave proactively will have an edge, because it is aware of the situation that is about to unfold as well as what has already unfolded. Anticipating (partial) future situations can be done when agents on a team can associate themselves with roles in particular team plans (predefined and/or scripted "plays"), since an agent in such a plan knows what others will be trying to do.

There are many sources of such plays when it comes to soccer. The rich knowledge of formations, plays, and strategies would appear to be important to tap into, and thus our agent design strategy has prefered an implementation that allows us to encode this knowledge directly, rather than by using learning mechanisms to get the agents to learn this knowledge inductively. Also, as new strategies are concocted or discovered, the agents' architecture should allow these to be easily and effortlessly incorporated with the pre-existing strategies.

Competitors for our team are likely to pursue particular team strategies of varying degrees of complexity, whether these are represented explicitly or captured implicitly in the configuration of the reactive agent behaviors. To the degree to which we can represent such strategies, our agents should be able to use these explicit models to construct evidential reasoning networks that support plan recognition. Again, our assumption is that all agents will be able to react to the current situation; agents that can predict likely emerging situations by anticipating their opponents moves will have an edge.

Finally, as strategies, methods, and plays are added to an agent, the agent itself should be able to select the most appropriate plan for its current/predicted situation, and when more than one applies equally to the situation, the agent should dynamically learn which seems to be the most successful in the current game. In this manner, an agent team can adapt to its opponent's strategy, strengths, and weaknesses.

From these assumptions, we are building a real-time multi-agent system that satisfies the following requirements:

1. The agents can follow simple reactive behaviors when these seem more appropriate.
2. The agent team can move flexibly between different levels of teamwork organization, from

each agent reactively working alone up to clearly articulated, precise multi-agent procedures.

3. The system allows the addition of new procedures, which will be selected over existing procedures on the basis of the goals they accomplish and the contexts in which they are appropriate.
4. The system can adapt by learning based on experience which actions are successful and which are not.
5. The set of available procedures available can be used (possibly after a representational transformation) for plan recognition to predict opponents' actions.

1.1.Reactive vs. Plan-Driven

Several approaches in the Pre-Robocup implemented a mainly reactive system, in which collaboration between agents emerged and was driven by present events. This approach can be very effective in the field of soccer, because soccer is highly dynamic, and it is usually not easy to carry through complex plans to completion (contexts are likely to change while executing a plan). Hence, most of the multi-agent strategies and interactions (collaboration and competition) have been limited to selecting the most immediately beneficial action(s) (e.g., the best pass in the present situation), and collaboration strategies (to the extent they have been used) have generally been encoded implicitly in the maintenance of team formations and roles.

Another problem with these mostly low-level reactive approaches is that these agents are often written in languages such as C which, while fast and efficient, renders the agents difficult to modify, since they often encode goals, subgoaling, and decision mechanisms implicitly in the code (with function calls, IF statements, etc.).

Our approach tries to overcome these limitations by using a procedural reasoning system (UM-PRS) which is able to deal with a dynamic and changing environment while still providing a general decision cycle in which goals and separate procedures can be added modularly. This approach allows more complex plans to be easily specified and selected, while the architectures emphasis on continuously checking against contexts to decide if the plan being executed is still valid supports reactivity. Consequently, collaboration between agents can use, on top of a reactive system, a set of shared plans which provide advantages that a simple reactive multi-agent system would not have.

1.2.Advantages of shared coordinating plans over reactive interactions

As we mentioned earlier, agents that share a set of coordinating plans on top of a reactive architecture system can have definite advantages over agents that are solely reactive. The advantages might be more apparent in domains (such as football or chess) which make extensive use of more standardized plans and strategies. Nonetheless, such plans are also present in soccer, but are usually hidden (implicitly encoded) in the team formation and player roles. Implicit encoding is not necessarily a good solution, since revising embedded strategies during a game is a complex task, judicious reasoning over strategies is more difficult, and agents using embedded strategies are not any harder to anticipate than those with explicit strategies, since opponents only see the externally manifested behavior anyway.

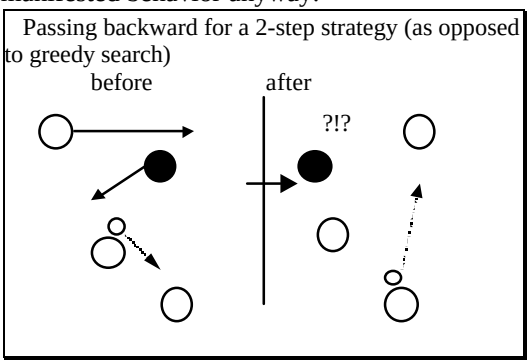

Figure 1

From a strategic standpoint, possessing and reasoning over shared plans can provide significant advantages. For instance, a local best action in the current situation is not necessarily the best choice in the overall game. In the case of a soccer competition, for example, an attacker might pass the ball backward (which is a step back when considering the immediate advantages) but profits from this action in the long run (e.g., disorienting the opponent defenders, see figure 1 where white wants to move the ball toward the top of the page ultimately).

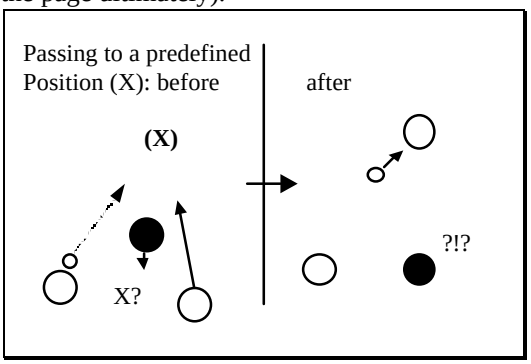

Figure 2

Furthermore, in an adversarial situation, shared plans among players of the same team allows for more subtle strategies; for instance, a player 1 in team A can pass to location X knowing that player 2 in team A will start to move to X to get the pass, but team B does not know about X and the plan of player 1 and 2, and since player 2 is not yet at X, team B will not be able

to predict the pass (or will wrongly predict a direct pass) and intercept it (see Figure 2).

2. Collaboration

2.1.Hierarchical levels of teamwork organization

A problem in using complex plans to coordinate teamwork is that complex plays rarely get completed undisturbed in a dynamic adversarial environment such as soccer. Also, it is possible that in given situations a more reactive (opportunistic) behavior is preferable to a complex coordinated action (e.g., a player is in front of an empty goal with the ball, and should try to score instead of passing as suggested by the currently used play). Consequentially, an agent should be able to behave according to complex plans and team strategies, as well as more basic reactive rules and behaviors that might not be directly specified in a complex plan.

In our implementation, we recognized a few levels of collaboration and team strategies, from the most basic, individualistic, and purely reactive actions, to more complex, team-oriented, and anticipatory plays. The agents are then able to select the appropriate level depending on context and on the amount of knowledge available.

2.1.1.Roles

A first simplified way of coordinating agents is to assign them roles based on player numbers or current positions, so that a team has a goalie, a given number of defenders, etc. Obvious roles are goalie, defender, midfielder, and attacker. Roles, however, are typically very static and individual: Given its role, an agent knows what reactions it should prefer and what its general responsibilities are. It is problematic for players to change roles (e.g. a midfielder acting like a defender or an attacker under some circumstances) because doing so implies a broader model of others' roles and current actions. When agents are given flexibility to change roles, breakdowns in team performance can occur (some roles are not fulfilled). When agents adhere to their roles and react appropriately within the bounds of the roles, their reactions are determined by the situation and thus can become predictable to opponents. Because they only know the roles of teammates but not where those teammates are likely to be fulfilling their roles, agents cannot anticipate the status of teammates not within the visual field.

2.1.2 Formations

Many of the conventional plays in soccer can be though of as being encoded within a set of well recognized formation (e.g., 4-2-4, 4-3-3, etc.), in which everyone in the team might be expected to remain in their respective assigned area of action unless the ball is within reach. In this context, passes can be directed to areas where other teammates are expected to be, without necessarily having to turn to that direction to make sure that a receiver is there and ready to receive the ball. Formations can offer only a limited amount of different plays (which are somewhat restricted, see figure 3), and are very predictable (they do not hide much from opponents' observations).

Formations allow agents to be "aware" of features of their environments that they might not be able to currently sense, because they can depend on teammates to be in particular regions. This is a step up from roles, but if agents are using this information to only look ahead a single action, the advantages are limited.

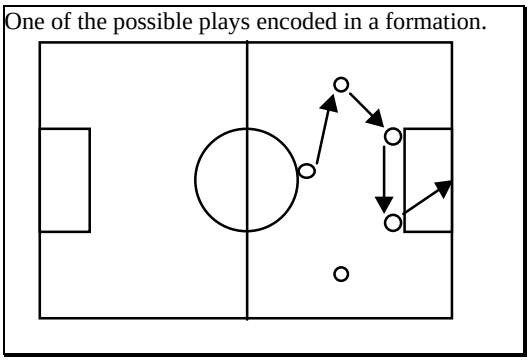

Figure 3

2.1.3.Plans

The next step is then to allow for any general play to be described (using UM-PRS in our implementation), which overcomes the problems mentioned earlier as well as having additional strategic advantages. Two very simple plans are illustrated in figures 1 and 2, but soccer has a rich domain of successful plays (e.g., restart plays) ready to be used.

In our UM-PRS implementation, unless a plan is applicable, an agent's behavior will default back into formation plays, simple roles, or a purely ball-driven behavior. The expectation is that knowledge-lean agents without defined plans will not be paralyzed, but rather will degenerate into a more reactive team. When opportunity for an established play presents itself, the agents can capitalize on it. This has similarities to how it appears many soccer teams play.

2.2.Support for collaboration

Plan contexts and the completeness of available information (see section 3) guide the selection between plans, strategies, and roles. However, we have developed additional protocols of communication and conventions among agents to support coordination and teamwork.

2.1.4.Location

A Cartesian coordinate map representation of the soccer field and of the agents' current positions provides an essential basis for collaboration. It allows agents to communicate absolute positions of objects on

the field, and to recognize and assume given formations based on their role (e.g., from conventional soccer formations such as 4-2-4, to simple role behaviors based on position, such as a goalie that stays within its own goal area, to agreed positions and targets, as seen in figure 2).

The current x and y position of an agent is derived from the visual information of objects such as flags, lines and goals, whose position is already known (using trigonometry, triangulation, etc.). Additional heuristics are necessary to deal with the possibility of incomplete information (due to position and restricted visual field), and to deal with the trade-off between correctness of estimate and time (or moves) required to obtain a correct value.

The position of any other object seen can be easily computed by using the current position of the agent as a reference. Also, other useful information that is easy to compute includes abstractions on location, such as 'close' or 'closest to ball', strategic areas (rather than points) such as 'left-field' or 'defense', and even more dynamic areas such as 'open-space' areas (e.g., holes in the opponent's defense), etc.

2.1.5.Verbal Communication

The Robocup server allows agents to communicate verbally, but agents are faced with several problems when using verbal communication:

1. Messages are unauthentic, so opponents can take incoming messages from the other team, scramble them, and re-send them to confuse adversaries. This can be addressed by using parity bits for messages (or other coding techniques), and by recording the direction of incoming messages and comparing it to knowledge of the relative direction of teammates and opponents.
2. Messages can be heard only within a certain distance radius.
3. Messages are not heard instantaneously (so that a chain propagation often takes too long, and messages are often obsolete before they even reach their destinations).

2.1.6.Communication and levels of belief:

For the above reasons, the agent should believe different information at different levels of certainty, such as information based on observations within visual field, information based on messages, potentially outdated information acquired previously, and estimated information based on inferences over plans and formations.. This hierarchy helps resolve conflicts and allows retraction of wrong information. Agents can communicate to complete each other's internal maps, coordinate group action, or even communicate beliefs on other agents' intentions (e.g., the opponent is planning to attack).

However, such communication should be used parsimoniously and cautiously, since messages take time to be communicated and understood (and might be outdated by then) and since the potential number of messages can quickly grow out of proportion. Another problem is that several agents could be sending conflicting messages. Most of these problems can be solved by deciding which agent should send which message depending on context (e.g., an agent should send messages on the part of the world to which it is closest to or most confident of, or a call to initiate group action is left to a leader or to the agent which is most obviously in the position to initiate the action (e.g., whoever has the ball, etc.)) and limiting messages to the most important or the ones which will still be useful in the near future.

Having said all this, messages can nevertheless be very useful as supporting information (when no other information is available), or to coordinate on a team strategy, etc. A few examples of use of messages are the following:

Calling plays for coordinated action
Calling about availability
Sharing information (e.g., absolute ball position)

3. An Architecture for Real-Time Agent Teamwork

3.1. UM-PRS

The underlying real-time architecture that drives our Robocup agents is UM-PRS (Lee et al, 1994). UMPRS is based on the SRI Procedural Reasoning System (PRS) developed by Georgeff et al, (Georgeff & Lansky, 1986). UMPRS is well suited for use in an agent architecture because it allows the agent to pursue long- term goals by adopting predetermined procedures based on context and not blindly following prearranged plans. It is also well suited for agents in a reactive environment where it can switch out its currently executing procedure and invoke another appropriate procedure when the situational context changes.

UMPRS is composed of five components, as seen in Figure 4:

Database -

A database or world model containing the beliefs and facts about the world.

Knowledge Areas (KA) -

A set of declarative procedure specifications that describe how to achieve a system goal or query. A KA consists of a purpose (a goal, test or query or action) for executing the KA. The context in which the KA is applicable and a body, which is viewed as a directed graph in which nodes represent states in the world and arcs, represent actions or subgoals. The body may consist of goals to achieve or maintain, goals to test, subgoals, branches, assertion or retraction of beliefs or primitive functions to be executed. This is just a partial enumeration of the set of actions. For a complete description see (Huber et al).

Goals -

UMPRS maintains a set of current goals to be realized, and KAs may place additional goals or sub-goals into the system by attempting to achieve some action.

Intention Structure -

The intention structure acts as the run-time stack of the system. It keeps track of the progress of each high-level goal and all of the subgoals. The intention structure suspends, resumes, cancels and proceeds with the execution of goals.

Interpreter -

The interpreter is what controls the execution of the entire system. Whenever there is new or changed information in the world model or goal list, the interpreter determines a new set of applicable KAs, places them in the intention structure and executes the next action.

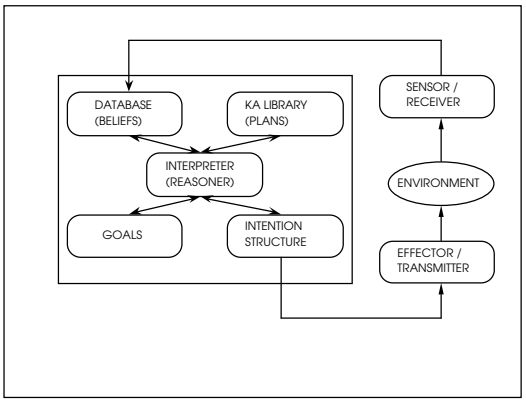

Figure 4 UMPRS architecture

3.1.Selecting among multiple actions and strategies

Because of the complexity of the robotic soccer domain, there are often several possible ways to achieve immediate goals or execute actions. So an advantage of using the PRS architecture is that it provides an automated mechanism for integrating these together and selecting among them depending on contexts, goals and preconditions.

For instance, our implementation can decide between passing (or moving) to a specific area and passing to a specific object, where one is preferable to the other depending on the context. Passing to an absolute x and y coordinate, for example, might be more risky, because it assumes that there is a teammate around that position and ready to receive the pass (and might shoot before looking), but has the advantage of potentially saving precious time by avoiding having to first turn to face the direction of the actual potential receiver. However, if the agent is already facing the right direction (so that the receiver of the pass is in sight), it can be preferable to pass to the receiver's direction and distance.

3.2.Actions and underlying UMPRS structure

The implementation of the UMPRS soccer agents is divided between C primitive functions and KAs, or procedures. The main distinction between the two is that primitive functions are usually atomic actions (such as parsing networking messages, triangulating locations, and sending individual commands such as turn, dash, say, etc.). KAs, on the other hand, are usually more complex and non-atomic actions or plans, where it is in the best interest to take advantage of the UM-PRS real-time selection of plans (KAs). For instance, a KA might specify to kick the ball to the side and dash after it when an opponent is moving toward us. There might be more than one way to move to the side (e.g., left or right), and depending on the presence of opponents to either side, one might be preferable to the other. UMPRS will take care of all these possibilities automatically, so that they do not overcomplicate the specification of simple procedures. (It also allows for easily adding yet a third way to move to the side without knowing of all the details of the other two.) Also, if at any time while executing a plan UMPRS finds that its context is no longer met, it will interrupt the execution of that plan and pursue a more appropriate plan.

It is clearly possible to implement anything that UMPRS accomplishes in another language or architecture (e.g., C or C++), but such an implementation will likely be very hard to modify and augment, and probably contain numerous if statements and flags embedded all through the code. UMPRS's separation of KAs from the selection and execution cycle avoids such problems and also presents additional advantages, as discussed in sections 5 and 6.

4. Preliminary tests

So far we have implemented the following:

A set of primitive functions which enable an agent to estimate its position in coordinates, and thus the coordinates of any object in sight, to check if a trajectory or path is clear of enemies, and adjust the estimate of moving objects in the next time step.

We have then implemented simple behaviors in the form of KAs, such as chasing the ball, staying within a specific area, dribbling (kicking the ball a short distance away and following it), passing (or moving) to a specific object or coordinate, etc.

Building upon these KAs and primitive functions, we have then implemented simple role behaviors such as goalie, defender, midfielder, and attacker. Agents would determine their roles based on their player numbers, but since their roles are based on a variable assignment, a role identity can be easily changed by a plan or by a context, when necessary. An example of a role behavior is a goalie, who stays within the goal area unless the ball is within reach, in which case it chases it and shoots it to a clear area (pass to a far

away coordinate using a trajectory which is unobstructed by enemies). Other roles such as the midfielder will share their area of action (the midfield) and attempt to pass directly to attackers whenever possible.

We have then added roles to implement formations (common soccer formations such as 4-2-4, 4-3-3, etc.), so that if the team is believed to be in a given formation, each agent will attempt to move to or stay within the area specified (which is not shared with other teammates, but rather is a partition of the entire field), and decide which is the next best pass given the formation and the trajectories which are unobstructed. Also, it can assume that other players are behaving according to formation, and therefore can pass to a given area and expect the respective teammate to be ready to receive the pass (this can save time because agents do not need to communicate, or even see each other).

We have then implemented very simple plans, such as the two shown in figures 1 and 2. In such plans, players would move to predetermined positions, and would check for specific conditions before going to the next step of the plan (e.g., receiving a specific message or seeing another player in a given position). These plans often take advantage of previously defined simpler KAs and roles, and UMPRS allows the agent to break from the plan when a context is no longer true (e.g., someone shouts "retreat!" or the opponent seems to have intercepted the ball, etc.)

5. Learning

5.1. Learning what works in a given context

In UMPRS, goals are broken down into subgoals, and contexts are used to find the most appropriate action that achieves that goal or subgoal. However, more than one action can apply to the same context and satisfy the same goal. So for instance, if an agent needs to avoid collision, it can accomplish this goal by either deviating to the left or to the right, both of which could be equally successful. When this happens, UMPRS uses both satisfaction of contexts and priority values given to actions and procedures to decide what to do. These priority values can be modified at runtime to allow the agent to adapt by learning which action is preferable from experience (by observing its outcome).

5.2.Finding appropriate learning feedback.

It is clear that in order to successfully learn in any domain, we need to have some feedback of whether a given strategy or choice was successful. This choice is very important but also very complex, especially in a domain such as soccer (how can we determine if any given action, e.g., passing, was really preferable?).

However, even a poor heuristic might be better than nothing. For instance, we might change our general strategy if we do not score within a certain time (clearly poor heuristic), or, better yet, if our immediate goal was not successfully reached (e.g., we attempted to pass, then after having kicked the ball we check whether the ball and the player we intended to get the ball are within a given distance from each other within a given time).

Once positive or negative feedback is gathered for an action (if it is gathered at all), it can be used to reorder a set of weights on alternative procedures which satisfy the same goal (say, evading to the left or to the right or at random). If heuristics for judging success of team plans are available, priority changes for the respective plans might have to be broadcast and propagated through messages passed among players.

In this way, we can achieve adaptation to a specific opponent and find the best choice of strategies that work, as well as become less predictable (even if the opponent learns our strategy, this will change once it becomes less successful). Also, this is often preferable to random selection of actions because actions and plans are chosen within a set of valid and effective actions, and because if the opponent has consistent behavior, once a good choice of actions has been found, it is desirable to keep using them, instead of relying on random selection.

6. Opponent modeling

Plan Recognition

Yet another advantage of using UMPRS as a real-time architecture for Robocup is that UMPRS itself can be extended to automatically generate bayesian networks for plan recognition [Huber 96] of any given agent (using observations and the assumption that every agent can be modeled with a similar set of methods, actions, and plans), which can allow for both improved agent collaboration and ability to model and predict the goals and actions of opponents. We are currently trying to incorporate plan recognition in our agents' implementation, to allow them to predict or estimate the goals and strategies of both teammates and opponents.

7. Conclusion

Our implementation allows an agent to choose the best strategy depending on the context and the amount or completeness of the information available. Furthermore, it allows us as developers to easily specify additional team plans which are automatically integrated with the pre-existing plans and actions, since goals and contexts are used to select the appropriate action or plan at runtime. Also, the UMPRS architecture appears to offer a straight-forward method to perform plan recognition and adaptive learning, which we hope to exploit.

So far, we have only implemented a few very simple team coordinating plans and plays, but this

initial implementation is able to successfully integrate reactive behaviors (chasing the ball, performing simple obstacle avoidance, etc.), roles (goalie, defender, etc.), formations (staying in formation, e.g., 4-3-3, and take advantage of assumptions about other teammates' positions and behaviors, and using the next best pass given the formation), and more complex plays (which may consist of several steps and are not obvious to the opponent).

Our next objective is to add simple heuristic feedback for actions and plays to allow the adaptation learning described in section 5, and to use the automated creation of belief networks mentioned in section 6 to allow plan recognition for coordination and for guiding the selection of appropriate plays and strategies.

Bibliography

[Lee et al] Jaeho Lee, Marcus J. Huber, Edmund H. Durfee, and Patrick G. Kenny, UM-PRS: an implementation of the procedural reasoning system for multirobot applications. In Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS '94), pages 842-849, Huston, Texas, March 1994.

[Huber et al] Marcus J. Huber, Jaeho Lee, Patrick G. Kenny, and Edmund H. Durfee, UM-PRS V3.0 Programmer and Refernece Guide, University of Michigan AI Lab Document, 1994

[Huber 96] Marcus J. Huber, Ph.D. Thesis, The University of Michigan, Ann Arbor, MI, 1996

[Georgeff, Lasky] Michael P. Georgeff and Amy L. Lansky. Procedural Knowledge. Proceedings of the IEEE, 74(10):1383-1398, October 1986.