Interleaving Deliberative and Reactive Planning in Dynamic Multi-Agent Domains

Rune M. Jensen
Department of Automation
Technical University of Denmark
2800 Lyngby, Denmark
ex04@iau.dtu.dk

Manuela M. Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3891, U.S.A.
mmv@cs.cmu.edu

Abstract

Reactive planning, consisting of pre-defined sensor-action rules, is well suited to effectively respond to dynamic changes in real-time environments. However, it is in general challenging to strategically reason about long or short-term objectives using reactive planning. Therefore, ideally, deliberative and reactive planning should be integrated. In this paper, we introduce an adaptive interleaving of deliberative and reactive planning as our approach for dealing with real-time dynamic environments. Two main aspects are responsible for the success of the approach. First, the deliberative planner uses depth-bounded forward chaining guided by goal-based heuristics. Second, the real-time state space is discretized as a function of the average time that the deliberative planner needs to generate a plan. This ensures that the state, as seen by the deliberative planner, does not change in average while the plan is being generated. When a plan fails or a new plan is needed, the reactive planner takes over. We extend our approach to multi-agent real-time domains, where the need for collaborative deliberative planning is particularly needed. We implemented and demonstrate our integration using the Prodigy deliberative planner and the RoboCup soccer simulator server.

1 Introduction

In one of Monty Phyton's sketches, ancient Greek philosophers play a soccer match. After the kickoff, the ball lies surprisingly untouched in the grass with the philosophers wandering scrutinizing about. Suddenly one of the philosophers shows an expression of great insight and initiates a sequence of precise passes among his team mates that finally places the ball in the opponent goal.

From an Artificial Intelligence (AI) point of view this sketch is amusing because it illustrates the problems of applying high-level reasoning systems like AI-planners to dynamic real-time domains. As implicated by the untouched ball due to the lack of actions to perform a major problem is the plan generation time. What should the agents do during plan generation? And how can we ensure that a generated plan is executable by the agents given the changes of their state during plan generation?

Another problem is indicated by the ironical success of the approach used by the Greek philosophers. In a real soccer match planning a goal from the kickoff to the scoring kick is impossible. Due to the dynamic changes of the environment (e.g., the opponent movements) the state (environmental conditions) assumed by a given plan only have a limited duration of consistency. Hence, an approach for integrating planning in these environments must be based on short limited plans. Further, even with this constraint, the plan execution is still vulnerable to failing during execution of plan step actions.

An alternative approach to deliberative plan generation followed by execution is real-time reactive planning, where the agent immediately responds reactively to dynamic changes in the environment. Re-

active planning can be viewed as consisting of a set of pre-defined sensor-action response rules. The advantage of this approach is that it allows short reaction time to environmental changes. The drawback is that the greedy strategy for applying actions resulting from only considering the current situation draws the agent towards local optimal states. Long or short-term objectives cannot easily be reasoned about and there is no immediate guarantee that a goal-based satisfying state is reachable. The focus on choosing an action in the current situation makes long-term behavior hard to represent and predict.

Consequently, the ideal approach for real-time dynamic domains seems to be a combination of classical deliberative and reactive planning. In this paper we present such an approach allowing the agent to execute plans when the environmental demand for response is low, and degenerate to reactive behavior when the environmental demand is high or a failure occurs.

Our approach has been inspired by MRR-planning introduced in [9]. In MRR-planning reactive and classical planners compete in each cycle for choosing the next action of the agent. If the response demand is low the action selection mechanism waits until all planners have finished processing making the agent behavior deliberative. Otherwise, if the response demand is high action proposals from reactive planners are weighted higher and an action is chosen as soon as its summed weight exceeds a threshold. Because the reactive planning is faster than the classical planning the agent's behavior in these situations becomes reactive with short response time.

In our approach we relax the requirement, that full-scale plans must be generated in each cycle in order to behave deliberative. Instead the agent iteratively executes time bounded plans. The plans are generated by a concurrently working forward chaining planning system, that repeatedly receives updated state information from the agent. Thus, as long the environmental demand is low, the agent can keep behaving deliberatively taking dynamic changes of the environment into account.

In each cycle the response demand of the environment is measured. If the demand is too high or the plan execution fails the agent degenerates to reactive behavior. Hence, the agent adapts dynamically to its own failures and changes of the environment overriding the planned actions.

In contrast to MRR-planning we further generalize our approach to multi-agent domains making collaboration between the agents an important goal of the deliberative planning. As an example domain, the RoboCup robotic soccer simulator [11] is used as a very challenging real-time dynamic environment. We further use the Prodigy deliberative planner [15].

The paper is organized as follows: section 2 discusses previous reasoning systems dealing with dynamic environments. In section 3 we introduce interleaving of reactive and deliberative planning for single agents in dynamic environments. In section 4 the approach is extended to multi-agent domains and demonstrated in the RoboSoccer multi-agent domain in section 5. Finally in section 6 we draw conclusions.

2 Previous Approaches

Several approaches have been pursued for the integration of planning and execution, which is related to the integration of deliberative and reactive planning. Our discussion of some approaches in this section is clearly not exhaustive, but we try to identify close work.

Most existing architectures for integrating deliberative planning in dynamic environments consist of a plan constructor and a plan executer interleaving passive planning phases and active plan execution phases. Recent systems in this category is the ROGUE architecture [7, 8] that integrates and learns from dynamic changes in the environment, and a system developed by Nourbakhsh [12] that uses conditional planning for dealing with environmental uncertainty.

As noted in the introduction, these approaches faces problems in domains with a high frequency of dynamic changes. Consider for example a flight control or a battle field scenario. Activity of agents during plan generation in these domains is crucial. Moreover, the agents must be able to respond instantly to changes in the environment during plan execution.

Architectures for addressing these problems have

been proposed by Lansky's *Procedural Reasoning System* (PRS) [6] and Brooks' subsumption architecture [2]. The subsumption architecture decomposes the behavior problem into task-achieving units realizing distinct behaviors of the agent separately. The units are ordered hierarchically with more complex behavior inhibiting simpler default behavior according to environmental demands. The subsumption architecture has proved to be efficient for implementing highly reactive behavior. Unfortunately the lack of high level problem-solving and lookahead makes the approach inflexible and hard to reuse in other environments.

The PRS architecture reasons about means and ends in much the same way as traditional planners, but further provides reactivity to dynamic changes of the environment. During execution a reasoner chooses different procedures (KAs), according to the state of the environment and the goals of the agent. If some important new fact or request becomes known, PRS will reassess its goals, and perhaps choose to skip the current active KA and work on another one.

Recently PRS has been applied to the RoboSoccer simulator domain [1]. Here KAs takes the form of simple *plays* among several agents (e.g. restart plays). In this way agents are capable of behaving deliberatively and perform actions not reactively optimal but efficient in the overall game (e.g., making a backward pass to confuse an opponent). Further, the PRS architecture allows agents to degenerate to reactive behavior, when the environment is demanding. A limitation of this approach though is the hand-coded nature of KAs. Agents should be able to develop new plays exploring the current situation using the available actions. In our approach the purpose of the depth bounded forward chaining planning is just to do that.

The *Agenda Based Control for Agents Behaviors Coordination* model (ABC^2) [10] also addresses the problem of combining classical and reactive planning. This model resembles PRS by having a central control process selecting *acts* from an agenda containing a set of acts currently under consideration. An act can either consist of processing or sending information to another agent or do some action, which may be requested by another agent. Like reactive systems the ABC^2 model is able to respond to dynamic changes, but in contrast to these systems ABC^2 also considers current goals of the agent when choosing actions. Unfortunately ABC^2 like PRS does not reason about future states. When applied to the RoboSoccer simulator domain this approach thus has much in common with elaborate reactive approaches, where collaboration is handled by role based formations [14].

An important difference between our approach and PRS and ABC^2 is, that we base reactive and deliberative planning on different representations of the domain allowing a long lookahead of the deliberative planning while keeping a low response time of the reactive planning.

3 Single-Agent Deliberative and Reactive Planning

Regardless of whether the agent uses deliberative or reactive planning it needs to process sensor information and to generate low level actions with a high frequency. Due to the fact that each planning mode uses different control principles and has direct access to sensors and actuators, the agent architecture can be described as a layered architecture (see Figure 1). The deliberative control layer controls the agent by executing plans generated by a concurrently working deliberative planner, while the reactive control layer controls the agent by executing actions generated by a reactive planner situated within the layer. In the single-agent case the deliberative planner can be integrated in the agent architecture. In our approach though the planner system must be an independent system in the multi-agent case, as the planner is shared by all of the agents.

In order to shift to deliberative planning a representation of the state of the agent must be estimated and sent to the planner system. The goal state is implicitly defined by a state evaluation function, and thus need not to be sent by the agent. To maintain the deliberative behavior we need repeated replanning with an updated state. The frequency of replanning depends on the stability of the environment. An unstable environment with a high frequency of

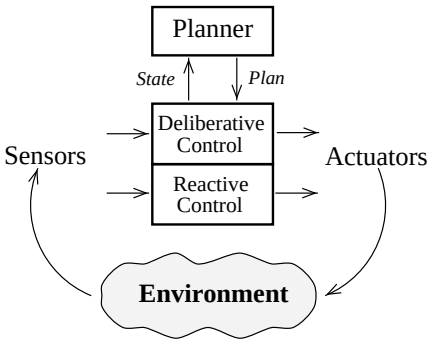


Figure 1: *Agent architecture.*

dynamic changes forces a short lookahead and consequently frequent re-planning. On the other hand a stable environment with only a few dynamic changes allows a high planning depth resulting in a longer lookahead.

3.1 Deliberative planning

In order to use a classical AI planner for reasoning in real-time domains a discrete state space representation of the domain is required. Finding the appropriate discretization is quite challenging. On one hand, the granularity of the discretization depends on which level of abstraction we would like to apply deliberative planning to. On the other hand, we would also like the state used by the deliberative planner to remain constant while the planner is generating a plan. Choosing an abstraction too fine can result in a long planning time and a high probability of not being able to apply the plan due to the state changes during planning.

To address this problem, we investigate a new method to determine an approximately correct granularity: *planner-dependent state discretization*. The key idea is that we discretize the state *as a function of the average planning time* required. We introduce a discrete planner state space S such that, for all the states $s \in S$, the average planning time is sufficiently less than the average time it takes a transition from s to occur.

Due to inevitable boundary conditions, this method is offers no absolute guarantee that the generated plan is still applicable in the state where execution should start. However empirical evidence in our implementation showed that the technique was suitable even for in our highly dynamic environment.

The planning problems we consider are characterized by having a specific initial state (e.g. the current agent position) and a loosely specified goal state as a disjunctive set of equally good positions on the field. Because this gives a one (current location) to many (several possible goal positions) search, we use forward chaining planning.

The operators used by the planner are classical STRIPS operators [5] and we use the Prodigy planner [15].

As the planning progresses we reason about the uncertainty of the plan being generated. The dynamic changes in the state cause regions of uncertainty to grow in the state space, that in turn inhibits further consistent planning. Hence, in our domain and planning algorithm, planning is depth bounded with a chosen depth, such that planning stops before it gets inconsistent.

The planning continues until the depth bound is reached. The achieved state is evaluated to determine if it is a goal state. If that is the case, then the corresponding plan is returned to the agent. Otherwise, the planner backtracks and finds another solution.

3.2 Reactive planning

In contrast to the deliberative planning the reactive planning is based on the current state without any lookahead. It uses a set of hand-coded sensor-action rules. The role of the reactive planning is to enable the agent to make fast responses to urgent demands of the environment and provide the agent with a basic behavior, that takes over if the deliberative planning fails.

Because the reactive planning is based solely on the current state it is defined as a decision tree, where nodes represent state conditions and leaves represent primitive actions.

3.3 Interleaving of deliberative and reactive planning

Due to the advantage of lookahead, the behavior of the agent is based on deliberative planning whenever it is possible. Thus when in a non-demanding situation the agent sends its state to the planning system and shifts to deliberative control as soon as the plan arrives (see Figure 2).

To make a continuous transition between the execution of two plans, the agent sends its new state just in time to receive the new plan before the old plan runs out. To raise the probability that the generated plan is applicable the new state is based on the assumption that the old plan succeeds. As noted in [1]

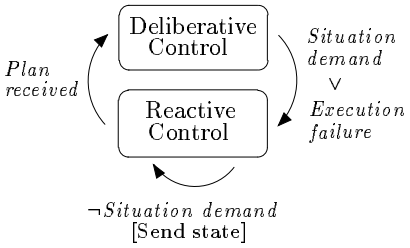


Figure 2: *Interleaving of deliberative and reactive planning for single-agent domains.*

an environmental demand can be both positive and negative. A positive demand is a sudden opportunity for the agent to explore the environment and reach a better state than possible by following its plan. On the other hand a negative demand is a problem not taken care of by the plan. In both cases the agent degenerates to reactive behavior. Further, a shift to reactive behavior also happens if the agent is unable to commit to the plan (see Figure 2).

4 Multi-Agent Deliberative and Reactive Planning

An extension of the single-agent architecture described above requires multi-agent planning and multi-agent plan execution. Current research in this field mainly deals with how to represent and share multi-agent plans in order to make distributed plan generation and negotiate which plan to follow [13, 4, 17]. We use one central planning system, which all agents connect to. The resulting architecture is shown in Figure 3. The multi-agent plan generation consists of three steps:

1. Fusion of possibly inconsistent and inaccurate world state observations from each agent to a consistent global world state defining the initial state of the planner.
2. Generation of a multi-agent plan.
3. Decomposition of the generated multi-agent plan to a coordinated single-agent plan for each agent.

In order to reduce the problem of coordinating the agents, their actions are synchronized by assuming all operators to take a fixed and equal amount of time. Further, to allow the planning system to handle an arbitrary number of agents, an operator only changes the state of a single agent. Nothing prevents

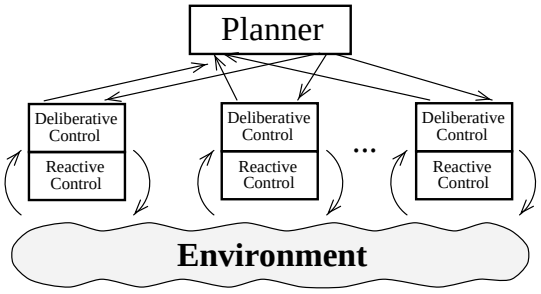


Figure 3: *Multi-agent architecture.*

the planning it-self to be asynchronous. But because operators for interactions between agents require synchronous state knowledge about the involved agents, we use synchronous planning. Thus, all agents must at least have reached time t before any operator, that leads an agent to time $t + 1$ can be applied. Consequently the depth bounded forward chaining planning takes the form of a simulation of the combined multi-agent state some number of steps ahead in time. The reactive planning used in the multi-agent extension of the architecture is similar to the

reactive planning used in the single-agent case. As argued by [14] roles are assigned to agents to enable efficient collaboration.

The last modification of the multi-agent extension concerns the interleaving of the deliberative and reactive planning. Due to synchronization of the agents during a deliberative phase the agents must enter this phase at the same time. But the agents should also exit the phase simultaneously, as the planned collaboration between the agents may be violated when one agent fails to commit to the plan. The agents achieve

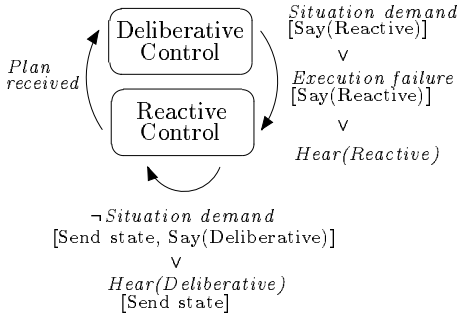


Figure 4: *Adaptive interleaving of deliberative and reactive planning for multi-agent domains.*

this by direct communication. When an agent cannot commit to the plan it communicates this to the other agents, which make a simultaneous degeneration to reactive behavior. On the other hand, when an agent finds the environment low demanding it communicates this to the other agents, which then try to enter a deliberative phase (see Figure 4).

5 Real-time Dynamic Multi-agent Domain: Robotic Soccer Simulator

Our example implementation uses an artificial environment provided by the RoboCup soccer server [11] and multi-agent plans generated by the nonlinear planner PRODIGY4.0 [3]. Each agent represents a player and is implemented as a separate program, that communicates with the soccer server and PRODIGY. The agent program generates an action command whenever it receives visual information send by the soccer server every 100 ms.

We base the deliberative planning on the position and direction of the ball and players. In order to obtain a discrete representation of the soccer domain we superimpose a grid to the soccer field. To reduce the branching factor of the search space we further require, that an operator can only make an agent move to an adjacent position.

As described in section 4 the synchronization of the agents during plan execution requires all planning operators to take a fixed amount of time. To facilitate the synchronization between agents during execution the transition between plan steps is based on the progression of time rather than the state of the agent. To ensure that the agents commit to the plan the agent state is monitored during plan execution.

The choice of the grid distance, operators and operator execution time has been based on multi-agent plan execution experiments and estimates of the corresponding planning time. Based on these experiments we have been able to define a set of operators that in addition to satisfying the timing constraints also is general by allowing many different plans to be defined and robust by having a high probability execution success.

The operator set contains four operators corresponding to the four combinations of possessing the ball before and after the execution of the operator:

`move <time> <agt> <x> <y>` The move operator is used to move an agent who does not possess the ball and who is not going to possess the ball at its next synchronization position. Similar to the remaining three operators the parameter set of the move operator contains the variables `<time>`, `<agt>`, `<x>` and `<y>`. The `<time>` variable specifies the next synchronization time, while `<x>` and `<y>` specify the next synchronization position of the agent `<agt>`. During the execution the agent position is monitored. If the agent at some point becomes unable to reach the goal position, the execution of the plan step fails.

`recv <time> <agt> <x> <y> <nextx> <nexty>` The recv operator is used to move agents who

do not possess the ball, but who are going to possess the ball at their next synchronization position. The parameters **<nextx>** and **<nexty>** specify the position the agent is supposed to move to after receiving the ball. The knowledge about this position allows the agent to position itself behind the ball directed towards the next position in order to increase the probability of a successful receive. The agent position is monitored during the execution of recv, which fails if the receive position at some point becomes unreachable.

drib <time> <agt> <x> <y> <nextx> <nexty>
The drib operator makes the agent dribble the ball. The agent thus possess the ball both at the current and the next synchronization position. Similar to the recv operator the parameters **<nextx>** and **<nexty>** specify the position the agent is supposed to move to during the execution of the next operator. The agent uses this information to alter the direction of the ball in the end of the execution of drib such that the ball is heading towards the next synchronization position. During the execution of drib, the agent kicks the ball in front of it and stays close to the ball. The execution of drib fails if the agent does not succeed to commit to the timing of these kicks.

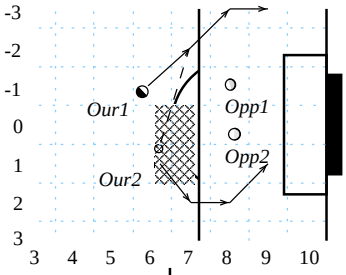
pass <time> <agt> <x> <y> <recvx> <recvy>
The pass operator makes the agent kick the ball to the position specified by **<recvx>** and **<recvy>**. The kick power is adjusted such that the ball arrives at the receive position at the next synchronization time. After the pass, the agent moves to its synchronization position. The execution of the pass fails if the agent does not succeed to kick the ball before some preset time limit or if the agent during the move phase becomes unable to reach the goal position at synchronization time.

PRODIGY4.0 is a domain-independent non-linear state-space planner. To generate multi-agent plans in the discrete soccer domain, we specified a forward chaining domain for PRODIGY including the operators above and the following literals:

```
(at <time> <agt> <x> <y> <dir>)
(has-ball <time> <agt>)
(init-pos <opp> <x> <y>)
```

where **<time>**, **<x>**, **<y>** and **<dir>** are discrete variables and **<agt>** and **<opp>** denote player agents and players from the opponent team. We only consider plans for attacks, i.e. the agents are assumed to possess the ball.

Consider the situation shown in Figure 5 (a). The



(a)

```
(at 0 our1 6 -1 -45)
(at 0 our2 6  1 -90)
(has-ball 0 our2)
(init-pos opp1 8 -1)
(init-pos opp2 8  0)
```

(b)

```
pass 1 our2 7 2 2 -2
recv 1 our1 7 -2 8 -3
drib 2 our1 8 -3 9 -3
move 2 our2 8 2
drib 3 our1 9 -3 10 -2
move 3 our2 9 1
```

(c)

Figure 5: *The plan generation sequence for generating plans of three steps for two agents.*

agents *Our1* and *Our2* send their state to the planner, which fuses the agent states to an initial state for the planning problem (Figure 5 (b)). The planner then generates a multi-agent plan of n steps (Figure

5 (c)), which is decomposed into single-agent plans and finally send back to the agents.

The application of operators is restricted such that agents cannot move outside the field or to a position occupied by a team mate. Further, only passes to a position where the ball can be received by a team mate are considered. Moreover, movements of opponents are modelled by defining a region around every opponent player that grows with time according to the average speed of opponents. Only moves and passes avoiding these regions are considered. We use control rules to guide the search towards a desired final state. Thus, the planning goal is implicitly stated by the control rules. Note though, that backtracking still occurs as the control rule guided forward chaining can lead to dead-ends, where no operators are applicable.

The timing diagram in figure 6 illustrates how plan generation and plan execution changes the low level action generation of an agent. In the diagram it is assumed, that the system consist of two agents. Thus, $state_i$ denotes the state send by agent i to Prodigy, while $plan_i$ denotes the plan received by agent i from prodigy. With a grid distance of 5 meters and an operator execution duration of 2000 ms PRODIGY is able to generate a multi-agent 3-step plan for two agents in less than 400 ms. As described in section 3.1 the application of the generated plan requires that the preconditions of the operators of the plan still hold at application time. In the soccer domain the main preconditions are, that the positions of the agents in the abstract planner space do not change during planning. As the average speed of the agents is 2.5 m/s, the average time for a position change of one agent is 2000 ms. For two agents the average time for a position change to occur is less than 2000 ms, but with a planning time less than 400 ms the probability that the generated plan is applicable is still very high.

In our example scenario we consider an attack by two agents interleaving deliberative and reactive planning against two defenders solely using reactive planning. The reactive planning used by the attackers consist of moving towards the ball if the ball is not possessed, and dribble towards the goal and eventually try to score if the ball is possessed. When dribbling the ball the agents try to avoid defenders by kicking the ball in the opposite direction of any defenders they see. Role based collaboration between the two attackers at the reactive level is restricted to avoiding the attackers to compete for the ball. When one attacker realizes, that the other attacker is nearer the ball it stops moving towards the ball, but keeps within a suitable passing distance. The defenders use a zone strategy to defend the right goal. Whenever the ball enters a circular region on the left and right side of the goal the agent protecting this zone moves towards the ball and tries to kick the ball to the left side of the field.

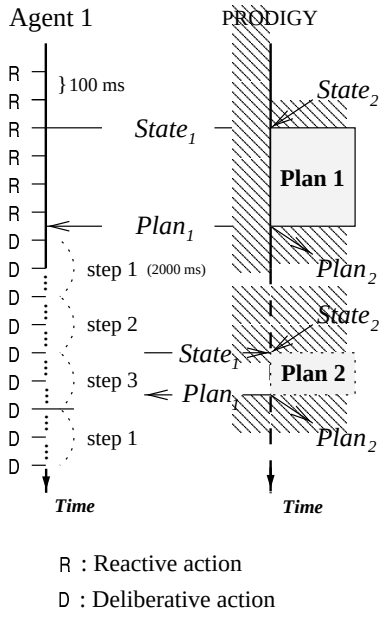


Figure 6: *Action generation timing diagram ($n_{step} = 3$ and $n_{agt} = 2$).*

Attackers shift from deliberative to reactive planning if defenders come too near (negative demand), the goal is score-able (positive demand) or the plan execution fails. When no defenders are nearby and the goal is unscore-able the attackers send their state information to PRODIGY and immediately starts executing the generated plan. At the beginning of the last plan step they send their new state information

to PRODIGY assuming that the current plan succeeds. Thus, the attackers will stay in a continuous phase of deliberative planning as long the demand from the environment is low and the plan execution succeeds. Figure 7 shows a plot of the positions of

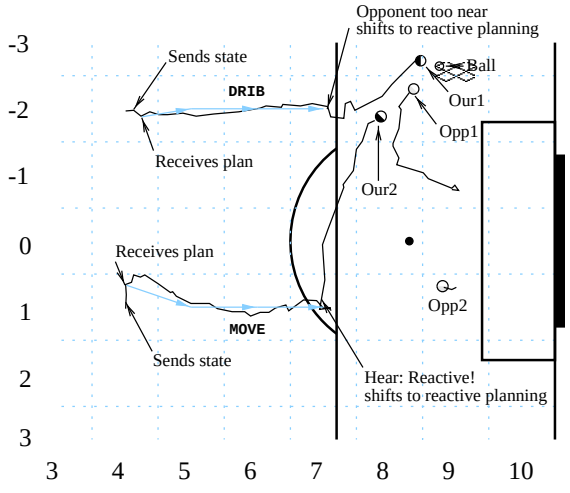


Figure 7: *Position plot of defenders and attackers during an attack.*

defenders and attackers during the first part of the execution of a real attack. Note how the reactive planning before the plan is received makes *Our2* move towards the ball. After the third plan step the situation becomes too demanding for *Our1* as *Opp1* is approaching. *Our1* communicates a shifts to reactive planning, which controls the rest of the actions. During the attack the ball only enters the defense zone of *Opp1*, thus *Opp2* does not move towards the ball. The experimental results show that deliberative and reactive planning can be integrated successfully.

6 Conclusion

In this paper we have shown, that integration of deliberative planning in real-time dynamic environments is feasible and efficient given an abstract representation of the domain. We introduce a planner-dependent approach for constructing a state abstraction used for the deliberative planning. Further, we describe adaptive interleaving of reactive and deliberative planning, that in contrast to other previous approaches (e.g. [6, 1, 10]) builds on different representations of the domain. The paper contributes to the current discussion on how to generate plans in dynamic domains by proposing frequent depth bounded re-planning as a means for integrating dynamic changes. We find this approach suitable for highly dynamic real-time domains like the soccer domain or a flight control domain. For domains with a low frequency of dynamic changes the price of re-planning may be too high. In these domains plan monitoring and elaboration may be more feasible [16].

Acknowledgments

This work was carried out while the first author was visiting Carnegie Mellon University. This research is sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement number F30602-97-2-0250. The views and conclusions contained herein 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 Research Laboratory (AFRL) or the U.S. Government.

References

[1] T. F. Bersano-Begey, P. G. Kenny, and E. H. Durfee. Multi-agent teamwork, adaptive learning, and adversarial planning in robocup using PRS architecture. *submitted to the Robocup 97 Workshop*, 1997.

[2] R. A. Brooks. A robust layered control system for a mobile robot. Technical Report 864, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1985.

[3] J. G. Carbonell, J. Blythe, O. Etzioni, Y. Gil, R. Joseph, D. Kahn, C. Knoblock, S. Minton, A. Pérez, S. Reilly, M. M. Veloso, and X. Wang. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Department of

Computer Science, Carnegie Mellon University, 1992.

[4] E. H. Durfee and V. R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, September 1991. Special Issue on Distributed Sensor Networks.

[5] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[6] M. P. Georgeff and A. L. Lansky. Rective reasoning and planning. In *Proceedings of AAAI*, pages 677–682, 1987.

[7] K. Z. Haigh. *Situation-Dependent Learning for Interleaved Planning and Robot Execution*. PhD thesis, School of Computer Science, Carnegie Mellon University, February 1998. CMU-CS-98-108.

[8] K. Z. Haigh and M. M. Veloso. Planning, execution and learning in a robotic agent. In R. Simmons, M. Veloso, and S. Smith, editors, *Proceedings Fourth International Conference on Artificial Intelligence Planning Systems*, pages 120–127. AIPS'98, AAAI Press, June 1998.

[9] S. Kurihara, S. Aoyago, and R. Onai. Adaptive selection of reactive/deliberate planning for the dynamic environment. In M. Boman and W. Velde, editors, *Multi-Agent Rationality*, volume 1237 of *Lecture Notes in Artificial Intelligence*, pages 112–127. 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'97, 1997.

[10] V. Matellán and D. Borrajo. Combining classical and reactive planning: the ABC^2 model. In R. Bergmann and A. Kott, editors, *AIPS'98 Workshop: Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, pages 121–126. AIPS'98, The AAAI Press, June 1998.

[11] I. Noda. Soccer server: a simulator of robocup. In *Proceedings of AI symposium '95*, pages 29–34. Japanese Society for Artificial Intelligence, December 1995.

[12] I. Nourbakhsh. *Interleaving Planning and Execution for Autonomous Robots*. PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, 1997. STAN-CS-TR-97-1593.

[13] A. E. F. Seghrouchni and S. Haddad. A recursive model for distributed planning. In M. Tokoro, editor, *Proceedings Second International Conference on Multi-Agent Systems*, pages 307–314. ICMAS-96, AAAI Press, 1996.

[14] P. Stone and M. M. Veloso. Task decomposition and dynamic role assignment for real-time strategic teamwork. *Third International Conference on Multi-Agent Systems (ICMAS'98)*, 1998.

[15] M. M. Veloso, J. Carbonell, M. A. Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[16] M. M. Veloso, M. E. Pollack, and M. T. Cox. Rationale-based monitoring for planning in dynamic domains. In R. Simmons, M. Veloso, and S. Smith, editors, *Proceedings Fourth International Conference on Artificial Intelligence Planning Systems*, pages 171–179. AIPS'98, AAAI Press, June 1998.

[17] M. Wooldridge and N. R. Jennings. Towards a theory of cooperative problem solving. In J. W. Perram and J. Muller, editors, *Distributed Software Agents and Applications*, volume 1069 of *Lecture Notes in Artificial Intelligence*, pages 40–53. 6th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'94, Springer-Verlag, 1994.