Reactive Behavior, Learning, and Anticipation

Steven D. Whitehead and Dana H. Ballard
The University of Rochester
Computer Science Department
Rochester, New York 14627

Abstract

When should a robot act and when should it think? Reactive systems always act, thinking only long enough to "look up" the action to execute. Traditional planning systems think a lot, and act only after generating fairly precise plans. Each represents an endpoint on a spectrum. When should a robot act and when should it think?

In this paper, it is suggested that this question is best addressed in the context of systems that learn. It is argued that primitive forms of reasoning, like anticipation, play an important role in reducing the cost of learning and that the decision to act or think should be based on the uncertainty associated with the utility of executing an action in a particular situation.

We present an architecture for an adaptable reactive system and show how it can be augmented with a simple anticipation mechanism that can substantially reduce the cost and time of learning.

1 Introduction

Much of the earliest research on intelligent robotics viewed robot control primarily as a two stage process of plan generation and execution. During plan generation the system reasons about the world in order to construct a plan that is followed during execution. Plan generation was considered the difficult, intelligent part of the problem; and received the most attention [13, 7, 15, 16, 18, 19, 5]. Plan execution, on the other hand, was considered more straightforward, less intelligent; and until recently received much less attention [6, 20]. Unfortunately, this two stage *reason then act* model for robot control has been shown to have several limitations, including: an inability to adequately model the effects of actions (variants of the frame problem)[10], an inability to adapt plans based on unexpected opportunities and contingencies (lack of flexibility)[1], and an inability to generate plans quickly (computational intractability) [5].

These problems have recently led to the emergence of *Reactive Systems*. These systems are primarily concerned with responsiveness and competence, emphasizing the *timely* generation of *appropriate* behavior in a wide variety of situations. Instead of relying on costly symbolic reasoning, reactive systems depend on precompiled knowledge about how to behave given a particular situation. Instead of following an explicit plan, generated by a planner, reactive systems use information present in the world (and sometimes a small amount of internal state), as an index to "look up" the action to execute next [1, 9, 8, 17, 2, 4, 12]. Another interesting property of *some* of these systems is that they contain no centralized, sequential controller. Instead, the parallel interaction

*This work was supported in part by NSF research grant no. DCR-8602958 and in part by NIH Public Health Service research grant no. R01 NS22407-01.

of a number of relatively simple, semi-independent subsystems leads to the emergence of intelligent behavior (Especially see [1, 3, 2]).

Reactive systems raise a number of important new issues. One issue is adaptability, which current reactive systems lack. Currently, knowledge about decision making is built into the system *a priori*. This precludes the system from exploiting unanticipated structure in the problem domain to improve its performance. Although machine learning has received considerable attention for quite some time, little work has been done to integrate machine learning results with reactive systems. A second issue is the relationship between reasoning and reacting[14]. From the standpoint of a cognitive model, reasoning is important. Introspectively, it is clear that humans contemplate the effects of actions, anticipate the future, and build and execute plans. It seems inevitable that highly intelligent robots must eventually incorporate mechanisms for anticipation and planning; but the role of reasoning in reactive systems is unclear. Current reactive systems do not include mechanisms for reasoning.

This paper claims that the role of reasoning in reactive systems is best addressed in the context of systems that learn. We argue that primitive forms of projection and anticipation may have evolved out of the need to minimize the cost and time associated with learning. Also, we define an adaptable reactive system that is based on the classifier systems described by Holland *et. al.*[11]. The system is massively parallel and relies on rule priorities to make decisions. Priorities are approximations of the expected utility of executing a rule's action. The system learns by adding and deleting rules and by adjusting rule priorities. Although the system may eventually learn to perform optimally, the time and cost of learning optimal behavior may be prohibitively expensive. To reduce this cost a simple projection mechanism is introduced. Projection allows the system to predict the effects of actions and to use those predictions to reduce uncertainty about the utility of executing an action. In familiar situations, projection is almost useless, but in novel situations it can greatly aid decision making. It is suggested that although primitive, this projection mechanism may be a precursor to more complex forms of reasoning.

2 Adaptable Reactive Systems

2.1 The Importance of Adaptability

Most reactive systems reported to date are static. The knowledge used for decision making is determined at design time and the performance of the system is the same after ten years of operation as it is on the first day. Although the current interest in reactive systems is important to the development of intelligent robotics, equally vital is that these systems be adaptive. There are several reasons why adaptability is so important. First, the world is extremely complex. Even in limited domains it is unlikely that a robot designer can anticipate all the subtleties that affect behavior. Second, the world is idiosyncratic. The world of each individual robot contains important features that can be exploited to optimize performance, but which cannot be anticipated in advance. For example, the spatial layout of a Mars rover's local environment cannot be anticipated, but can be learned and used to increase the system's performance. Third, the world is dynamic; components fail, objects move, objectives change. Without an ability to adapt, a once optimal system can become useless. Evidently, non-adaptive systems are doomed to be useful only in small, static domains.

2.2 Adaptable Reactive Systems

Recently, Holland *et al.* have proposed inductive mechanisms for adapting massively parallel rule based systems. They have used these mechanisms to model various forms of learning, from classical conditioning to scientific discovery[11]. One application of these elegant ideas is to adaptable reactive systems. By an adaptive reactive system we mean a massively parallel, adaptive rule based system

which controls a robot by repeatedly examining the current sensory inputs, and using its rules to reactively pick and execute an action. Formally, an adaptable reactive system (or ARS) is defined by the tuple $< S, C, E, D >$. $S = \{s_1, s_2, ..., s_n\}$ is a set of atomic propositions describing the current state of the world and $C = \{c_1, c_2, ..., c_l\}$ is the set of actions the system can execute. $E = \{e_1, e_2, ..., e_{2^n}\}$ is a set of state utility estimates. There is one utility estimate, e_m, for each world state, $m \in W$, where W is the set of world states. Intuitively, the utility of a state is a measure of the value of being in that state. Since in general the robot does not know the utility of a state *a priori*, the utility estimates are approximations to the actual state utilities. The estimates are incrementally improved using a technique known as the *bucket brigade* algorithm. $D = \{d_1, d_2, ..., d_k\}$ is a set of production rules of the form: $d_i : P_i \xrightarrow{\rho_i} G_i$, where P_i is a sentence constructed out of the atomic propositions S and the connectives $\wedge, \vee, \neg$ in the usual way, the action $G_i \in C$ is a single executable action, and ρ_i is a priority value used to guide decision making.

An ARS operates as follows. At each time step (or trial), the system is presented with an input pattern describing the state of the world. The pattern is used to evaluate the conditions (LHS) of the production rules. The set of rules whose conditions are satisfied on a given trial are said to be *active*. An active rule represents a tendency for the system to execute the action associated with that rule. For example, the rule: $s_1 \wedge s_2 \rightarrow c_1$ says "If $s_1 \wedge s_2$ is true in the current situation then *try* to execute action c_1." On each trial, multiple rules are likely to become active and several actions may be suggested for execution. The system is allowed to only execute one action per trial and uses a priority based bidding mechanism to choose the action to execute. The bidding mechanism works as follows: on each trial, an active rule is a bidder if there is no other active rule whose condition is more specific. Each bidder makes a bid for its associated action. A bid contains *an action* and *a priority*. After all the bids are in, the bids are tallied and a total bid for each action is determined. The total bids are then used to probabilistically choose the action to be executed. For example, one means of determining the action to execute is to assign a probability to each action on a per trial basis that equals the normalized sum of the bids received for that action. That is,

$$pr(a_i) = \frac{\sum_{j \in B_{a_i}} \rho_j}{\sum_{k \in B} \rho_k} \tag{1}$$

where $pr(a_i)$ is the probability that the system chooses to execute action a_i, B_{a_i} is the set of bidders who bid for action a_i, B is the set of all bidders, and ρ_j is the priority of bid j. A rule is said to have *executed* in a trial if it is active and its action is executed.

Clearly, rule priorities play a crucial role in determining system performance. How are priorities assigned? Utility estimates are used to determine rule priorities. The priority of a rule is adjusted to approximate the expected utility estimate of the state that results from *executing* the rule. That is,

$$\rho_i \approx \sum_{m \in W} e_m \cdot pr(m \mid d_i) \tag{2}$$

where $pr(m \mid d_i)$ is the probability that the system will be in state m at time $t+1$ given that it executes rule d_i at time t. A simple algorithm for computing a rule's priority is to think of the utility estimates as supplying a form of *internal reenforcement*. That is, in each trial the system executes an action and gives itself an internal reenforcement equal to the utility estimate of the resulting state. This reenforcement is used to incrementally adjust the priority according to the following rule:

$$\rho_i(t+1) = \begin{cases} \gamma e(t+1) + (1-\gamma)\rho_i(t) & \text{if rule } d_i \text{ is executed in trial } t \\ \rho_i(t) & \text{otherwise} \end{cases} \tag{3}$$

where $e(t+1)$ is the utility estimate of the resulting world state at time $t+1$ and γ is a constant that geometrically decreases the weight given to trials that occurred long ago. The basic architecture for an ARS that uses utility estimates for internal reenforcement is shown in Figure 1. Using this

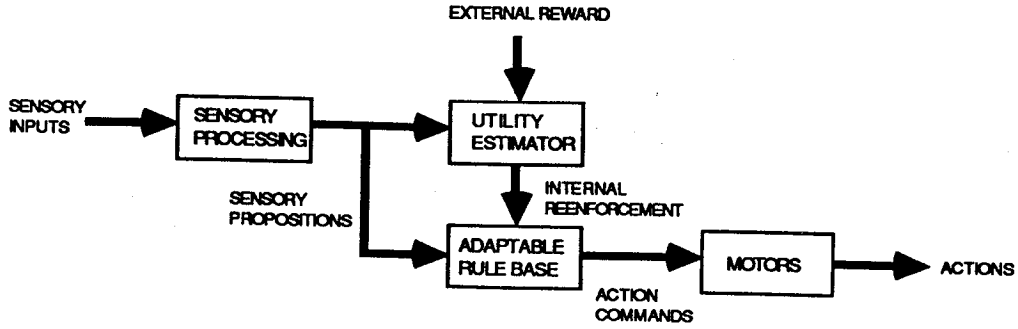

Figure 1: The basic architecture of an adaptable reactive system, (ARS).

priority assignment scheme, it is easy to see that if the system has good utility estimates it will tend to execute those actions that maximize the expected utility. The trick then is to define an appropriate utility function and a mechanism for learning it.

Although any number of utility functions can be defined, a good approach is to define the utility of a state to be the expected reward of possible futures that begin in that state. That is, suppose that an *external reward* is associated with each world state, and that the robot's objective is to obtain as much reward as possible. Intuitively, the robot wants to be in or "near" states that have a high reward associated with them, and the utility of a state should reflect this preference.

Formally, a *possible future* for the robot, p, is a sequence of states and actions, $m_0, a_0, m_1, a_1, \ldots$, where m_i is the state of the environment at time t_i and a_i is the action executed by the system at time t_i, respectively. Further the *utility of a possible future* is a weighted sum of the reward received by the system upon following that possible future, namely:

$$U_p = \sum_{i=0}^{\infty} \beta^i r(m_i) \tag{4}$$

where, U_p is the utility of possible future p, β is a constant used to geometrically diminish the importance of states on the path that are far into the future, and $r(m_i)$ is the external reward received by the system in state m_i.

The *utility of a state* is then defined as the expected utility of possible futures that begin in that state. That is,

$$U_m = \sum_{p \in F_m} U_p pr(p \mid m) \tag{5}$$

where U_m is the utility of state m, F_m is the set of possible futures that begins in state m, and $pr(p \mid m)$ is the probability that possible future p is followed given that the system starts in state m. Given this definition of utility, it can be seen that by maximizing its expected utility, a system also approximately maximizes its expected reward.

As mentioned above the system doesn't know the state utilities *a priori*. Instead utilities are learned as the system experiences the world and receives external rewards. At any given time, the utility estimates, E, represent the system's best approximation of the state utilities. Utility

estimates are incrementally adapted using the *bucket brigade* algorithm described by Holland *et. al.*[11]. The basic idea behind the bucket brigade algorithm is that state utilities are related by the following system of equations:

$$\forall_{m \in W}[u_m = r(m) + \sum_{m' \in W} u'_m pr(m' \mid m)] \tag{6}$$

where u_m is the utility of state m, and $pr(m' \mid m)$ is the probability that state m' is obtained at time $t+1$ given the system is in state m at time t. The bucket brigade algorithm makes use of these equations to incrementally build up approximations of a state's utility based on approximations of other states' utilities. Although the details of the bucket brigade algorithm are extremely interesting, for the purposes of this paper, it is sufficient to know that such an algorithm exists and that a massively parallel implementation is realizable.

In addition to estimating state utilities and modifying rule priorities, the system can adapt by generating new rules and replacing old "less useful" ones. Although the algorithms for this form of learning are important, they are not central to this paper and therefore will not be discussed. It is sufficient to say, that at any given time only a subset of possible rules are actually in the system (due to space constraints) and that as the system learns, rules are introduced and deleted in way that tends to keep around those rules that contribute most to the system's performance.

2.3 Stacking Blocks

To illustrate the ideas presented above, consider a robot that plays the following game. The robot sits in front of a table that has three blocks on it: A, B, and C. The robot can manipulate the blocks and arrange them in any configuration of stacks it likes. When the robot is happy with the arrangement of blocks it presses a small button near the edge of the table and receives a reward that is commensurate with the arrangement of blocks on the table. After the robot receives a reward, the blocks are randomly "scrambled" for another round. The robot's objective is to maximize its overall reward.

To formalize this game the sensory propositions S, the available actions A, and the reward function r must be defined. Suppose the robot has the following sensory propositions: $S = ON \cup CLEAR \cup LEVER$. Here ON is a set of propositions describing whether one block is on top of another or not:

$$ON = \{ON(A,B), ON(A,C), ON(A,Table), ON(B,A)...\}$$

where $ON(i,j)$ is true iff block i is on block j; $CLEAR$ is a set of propositions that describe whether or not a block has another block on top of it:

$$CLEAR = \{CLEAR(A), CLEAR(B), CLEAR(C)\};$$

and LEVER is a proposition that is true just when the robot has pushed the lever to receive the reward. Suppose further that the robot can execute the following actions: $A = PUSH\_LEVER \cup MOVE$, where $PUSH\_LEVER$ is an action for pushing the reward lever and $MOVE$ is a set of actions for placing blocks on top of each other and on the table:

$$MOVE = \{MOVE(A,B), MOVE(A,C), MOVE(A,Table), MOVE(B,A), ...\}$$

Here $MOVE(i,j)$ represents the action of placing block i on top of object j. Assume that both the block being moved and the target object must be clear for the action to successfully complete. Otherwise, assume the action fails and the configuration of blocks remains unchanged. Finally, suppose the reward function is defined as follows: If the reward lever is up (ie $LEVER = T$) then the reward is zero for all block arrangements. If the reward lever is down, then the reward value is

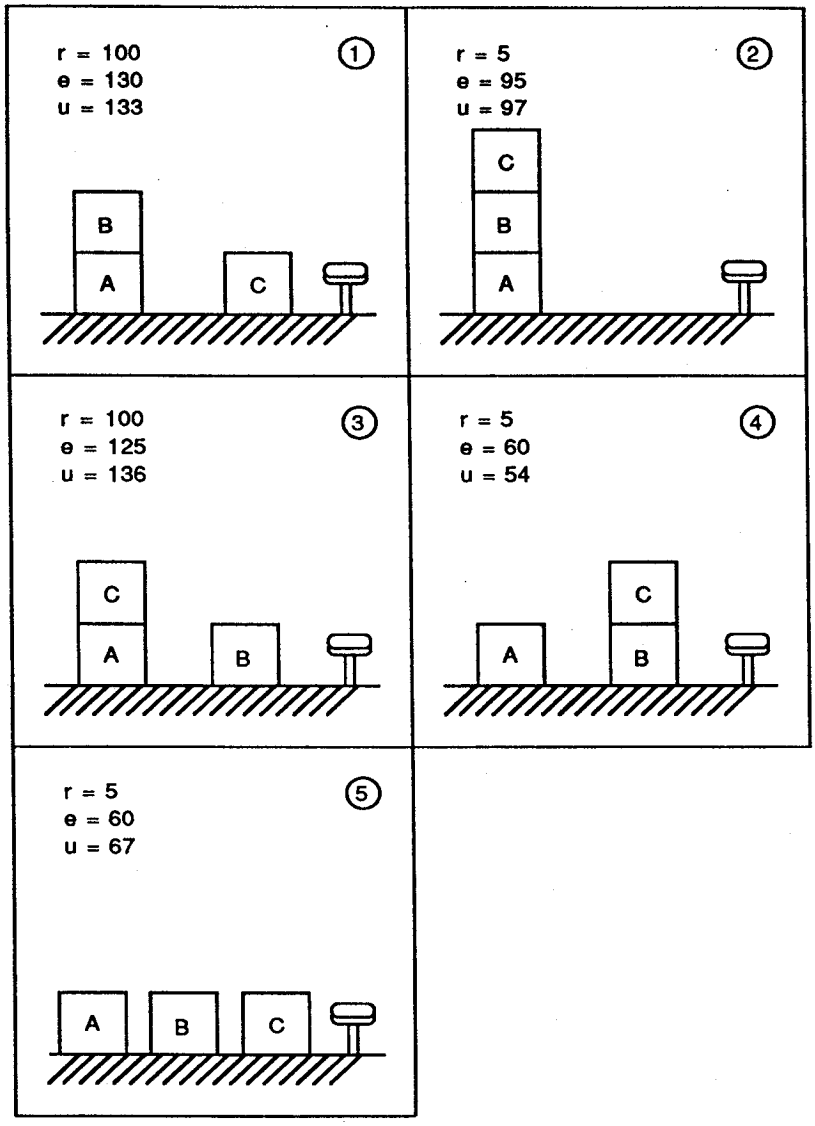

Figure 2: Five states of the block stacking game. In the upper left hand corner of each frame is the utility value and the utility estimate associated with the state. Also shown is the reward value associated with the state with the block configuration shown but with the reward lever depressed.

No.	Rule	ρ	States Active
1	$ON(B,A) \wedge ON(C,Table) \wedge ON(A,Table) \wedge \neg LEVER\_DOWN \rightarrow PUSH$	129	1
2	$ON(A,Table) \wedge ON(B,A) \wedge ON(C,B) \wedge \neg LEVER\_DOWN \rightarrow PUSH$	37	2
3	$ON(A,Table) \wedge ON(B,A) \wedge CLEAR(C) \wedge \neg ON(C,Table) \wedge \neg ON(C,A) \rightarrow MOVE(C,Table)$	109	2
4	$ON(B,Table) \wedge ON(A,Table) \wedge CLEAR(B) \wedge ON(C,A) \rightarrow PUSH$	132	3
5	$\neg ON(C,Table) \wedge CLEAR(C) \rightarrow MOVE(C,Table)$	75	2, 3, 4, +
6	$ON(A,Table) \wedge ON(B,Table) \rightarrow PUSH$	65	3, 4, 5, +
7	$CLEAR(A) \wedge CLEAR(C) \rightarrow MOVE(C,A)$	72	4, 5, +

Table 1: Typical rules from an ARS robot playing the block stacking game. Rules are numbered in column 1 and and shown in column 2. Rule priorities are shown in column 3 and the states in Figure 2 in which the rule is active are listed in column 4. The first four rules are specific, the last three are general.

100 if block A is on the table with another block on top of it, and the third block is on the table. Otherwise the reward is 5.

Figure 2 shows five frames each representing a possible world state. The utility value u, and the utility estimate e, for each state is shown in the upper left hand corner of each frame. The utility value is fixed when the game is defined but is unknown to the robot. The utility estimate is the robot's model of the utility function and changes as the robot plays the game, based on the bucket brigade algorithm. Also shown in the upper left hand corner is the reward value associated with the state with the block configuration shown but with the reward lever in the down position. The two states with the reward lever down and the block configurations shown in frames 1 and 3 are the only states where the robot receives a reward of 100.

Of the five states shown, states 1 and 3 have the highest utilities. Intuitively this follows since in these states all the robot has to do to receive a large reward is press the reward lever. The utilities of the states 2, 4, and 5 are somewhat less since the robot has to go further to receive a large reward. The utility estimates shown in the figure indicate that the system has a fairly good model of the utility function.

A partial list of rules for a robot playing the game is shown in Table 1. Although this list is not complete it is representative of the kinds of rules the system will generate. The first four rules are very specific. In each of these rules, the condition is satisfied in only one state (rule 1 is satisfied in state 1, rules 2 and 3 in state 2, and rule 3 in state 3). Rules 5 - 7 are more general and are satisfied in several states. Specific rules are important for encoding knowledge about the correct action to take in familiar situations, because they are satisfied only in very specific situations and are likely to have good utility estimates. Specific rules, however, are less useful for dealing with novel situations since they are less likely to be become active. General rules are the opposite; they are active more often but are less useful in familiar situations. The reason they are less useful in familiar situations is that the internal reenforcement they receive tends to be highly variable and their priorities are less likely to reflect the actual utility of executing an action *in a particular situation.* Figure 3 illustrates this point by showing that the priority of a specific rule is an average of utilities over a very small number of states while the priority of a general rule averages utilities over many states. In general, an ideal rule is one that 1) consistently receives the same reenforcement (low variability) and 2) is applicable in many states (abstract). Specific and general rules tend to have one or the other of these properties. As the robot plays the game it will tend to generate more and more specific rules as it learns about the state space. Initially, general rules are good since they represent heuristics or tendencies that lead to good payoffs. However, they are eventually replaced by more specific rules (with less variable reenforcement) that are particular to the idiosyncrasies of particular states.

To illustrate the basic operation of the system, suppose the world is currently in state 2. At the

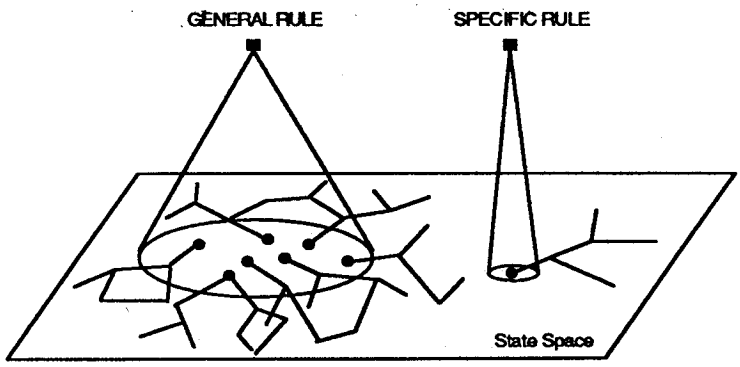

Figure 3: The priority of a general rule is the average utility of many possible futures since the rule is satisfied in may states. The priority of a specific rule, however, is an average over many fewer states and therefore is likely to have a smaller variance.

beginning of the trial, the sensory propositions are set and used to evaluation rule conditions. In state 2 rules 2, 3, and 5 are satisfied and become active. Of these rules, 2 and 3 are *most specific* and bid for control. Rule 5 is a generalization of rule 3 and therefore is a non-bidder. Since rules 2 and 3 call for different actions, the system probabilistically decides whether to execute $MOVE(C, Table)$ or $PUSH$. That is, according to Equation (1):

$$pr(MOVE(C, Table)) = \frac{\rho_3}{\rho_3 + \rho_2} = \frac{109}{109 + 37} = 0.75 \tag{7}$$

and

$$pr(PUSH) = \frac{\rho_2}{\rho_3 + \rho_2} = \frac{37}{109 + 37} = 0.25 \tag{8}$$

where $pr(MOVE(C, Table)$ and $pr(PUSH)$ are the probabilities of executing the move and push actions respectively.

Suppose, in this case that the system decides to execute $MOVE(C, Table)$. Then after moving C, the world is in state 1 and the priorities of rules 3 and 5 are updated based on the utility estimate for state 1, namely $e_1 = 130$. The priority of rule 5 is updated because its condition is satisfied and the action chosen was consistent with the action specified by the rule. Once the system is in state 1, based on rule 1, it is likely to push the reward lever and receive a sizable reenforcement.

3 Anticipation in Systems that Learn

3.1 Why anticipation is good

Almost all learning systems repeat the following cycle again and again: The system senses the input, decides on and executes an action, receives feedback based on the action, and adjusts itself to better its performance. The problem is that if the system learns slowly it must endure a lot of negative reenforcement. Learning is expensive. Using a faster learning algorithm is one way to reduce this cost, *anticipation* is another. Here, anticipation means the ability to predict future states by *projecting* the effects of actions. When a system without anticipation encounters a novel situation, where it doesn't have a good estimate of the utility of possible actions, it must randomly choose an action, suffer the consequences, learn and go on. In environments where wrong moves can be catastrophic, systems without anticipation are doomed to suffer. If a system can recognize that

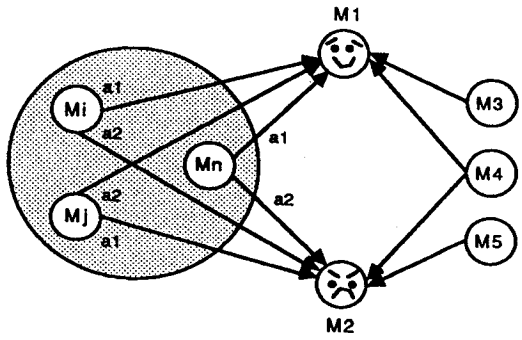

Figure 4: By entering states m_1 and m_2 from states m_3, m_4, and m_5 on previous trials, the system has learned that m_1 has high utility and that m_2 has low. Entering the new situation, m_n, a system without anticipation must rely on general rules which may be unreliable. A system with anticipation, however, can project the effects of possible actions and obtain better utility estimates, which result in better decisions.

it is in an unknown situation and project the effects of its actions, it may be able to obtain a better estimate of the utility of executing an action in that particular situation.

Figure 4 illustrates the idea. Suppose that it has been determined with good certainty that states m_1 and m_2 have very high and very low utilities respectively. The certainty of these estimates could be based on experience gained while visiting m_1 and m_2 from states m_3, m_4, and m_5. Further suppose the system has found itself in the novel state, m_n, for which it has only a rough estimates of the utilities of executing actions. These rough estimates come from "general" rules that are developed while executing in similar situations, like states m_i and m_j. Without anticipation, the system must rely on its general rules to guide its behavior. In this case, there is a good chance that the system will make a poor decision and suffer. If the system is capable of projecting the effects of actions, then it can "internally" explore its options by looking at the utility estimates of projected states, and feed back those estimates to *temporarily* adapt rule priorities. In the example above, the system might simulate executing action a_2 to find it yields state m_2; use the low utility estimate of m_2 to temporarily degrade the priority of the rules that caused a_2 to be executed; evaluate the rules again; simulate action a_1; find it leads to a high utility state; temporarily upgrade the priority of those rules; and finally execute a_1. Of course, a projection can be more than one step into the future. If the utilities of the projected states are themselves uncertain, the system can continue to project until it reaches a state that has a stable estimate.

In the context of the block stacking example given above, state 4 is a relatively novel state since only general rules are active in this state; in this case rules 5, 6 and 7. Notice how the priorities associated with each of these rules is roughly an average of the utility estimates of the states that are obtained after executing the rule any time its condition is satisfied. For any given activation of these rules, the actual internal reenforcement received may vary wildly. For example executing rule 5 in state 2 yields a reenforcement of 130, while executing rule 5 in state 4 yields a reenforcement of only 60. Using anticipation, the system can predict the low yield obtained by executing rule 5 in state 4 and can temporarily lower its priority. This in turn leads to the anticipation of the effect of rule 7, which yields a high internal reenforcement when executed in state 4. This finally leads to a temporary increase in the priority of rule 7, and eventually causes it to be executed.

As can be seen from the above examples, the variance in the internal reenforcement received by a rule is crucial for determining whether to act or anticipate. If the variance is low, the priority is a good approximation of the internal reenforcement the system is likely to receive when it executes the

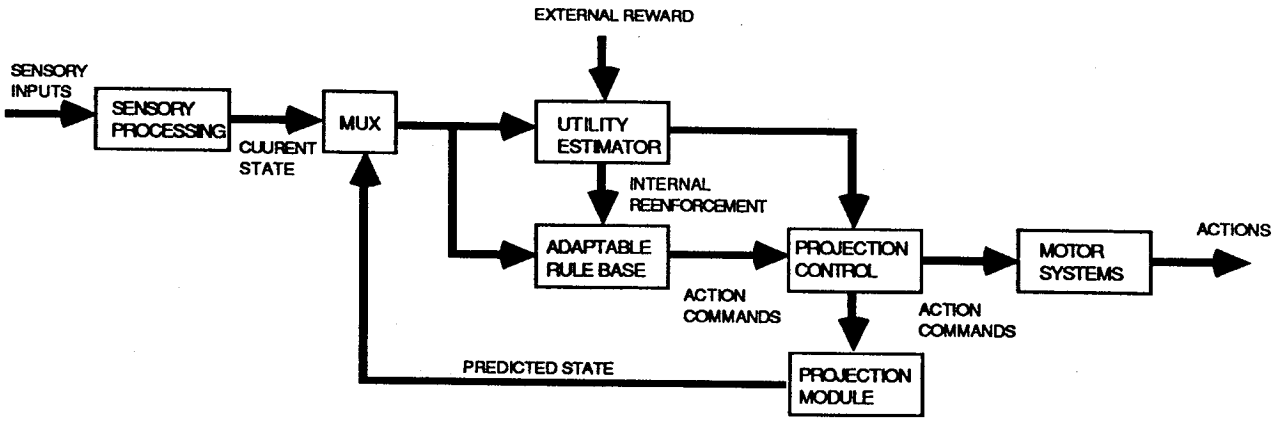

Figure 5: The ARS architecture augmented with a projection mechanism. In unfamiliar situations, where the uncertainty about the utility of an action is high, the system projects the effects of its actions to reduce uncertainty.

action. In this case, anticipation isn't very useful because it will just yield the same result. However if the variance is high, the actual reenforcement received may be very different than the average. In this case, anticipation is useful since it can lead to a much better approximation of the utility of executing the action in a particular situation.

3.2 Adding Anticipation to the ARS architecture

To add an anticipation mechanism to the ARS architecture, rule priorities are augmented with a variance value, σ_i, which is an estimate of the variance in the internal reenforcement received when the rule executes. Similarly, a variance value is also maintained for the utility estimates associated with each state.[1] Finally, a prediction mechanism is added that maps states and actions into predicted states.[2] The resulting architecture is shown in Figure 5.

The basic algorithms implemented by the architecture are given below:

Action Selection Algorithm

For each trial:

1. determine the active rules;
2. determine the bidders (most specific rules);
3. probabilistically pick an action to execute (based on current priorities and priority variances);
4. If the variance associated with the chosen action is low enough, say below V_thres then execute the action; otherwise:
   (a) project the effects of executing the action;

[1] The algorithm for determining utility variances is simple, and based on a slight modification of the Bucket Brigade algorithm.

[2] In general predictions must be learned as well. However for the purposes of this paper, we assume prediction knowledge is known *a priori*.

(b) use the results of the projection to temporarily adjust the priorities and variances of active rules, and

(c) go to Step 3;

Projection Algorithm

For each projection:

1. Based on the current world state and knowledge about the effects of actions, predict the state of the world after executing the action;
2. If the uncertainty about the prediction is too high, say above P_thres, then quit the projection and return a degraded priority value; otherwise,
3. If the variance of the utility of the predicted state is low enough, say below V_thres, then quit the projection and return the utility of the predicted state (degraded by β) and its variance as new estimates of the action's priority and variance respectively; otherwise,
4. Use the current predicted state as the basis for another projection.

There are two reasons to terminate a projection. First, if there is a large uncertainty about the results of the projection, then the projection may as well terminate because further projection probably won't help reduce uncertainty. Second, if the projection has lead to a state with a stable utility estimate (low variance), then the projection should terminate because this information can be used to lower the uncertainty and further projection is not likely to be more useful.

4 Summary and Future Work

In this paper, an adaptable reactive system was presented. It was shown that by adding a simple anticipation mechanism the cost of learning could be reduced by allowing the system to avoid undesirable situations. It was argued that the decision to act or project should be based on the uncertainty in the utility of executing an action in a particular state. Using anticipation this uncertainty is reduced by projecting into the future until a state with a stable utility is obtained. This utility is then used to derive a more certain estimate.

As of this writing, empirical results regarding the savings gained by adding projection are not available. We are currently in the process of implementing a system that plays the block stacking game. There are also several other directions for future work. One limitation of the current system is that the sensory and motor systems are binary. We hope to replace sensory propositions with semi-continuous variables, and augment the motor system to allow multiple simultaneous actions.

Abstraction is another area that needs attention. In the current system projections are one "primitive" action at a time; it would be interesting to develop an abstraction mechanism that allowed projection over "abstract" actions. Finally, short term memory has been ignored. It was assumed that the sensory information available to the system was sufficient to define the state of the world. This is fallacious in general and memory is necessary to overcome this simplification. It is also interesting to speculate about how memory could be coupled with projection to lead to more complex forms of reasoning.

References

[1] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *AAAI*, pages 268–272, 1987.

[2] James S. Albus. *Brains, behavior, and robotics.* BYTE Books, Petersborough NH, 1981.

[3] Rodney A. Brooks. Achieving artificial intelligence through building robots. AI Memo 899, MIT, 1986.

[4] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–22, April 1986.

[5] David Chapman. Planning for conjunctive goals. *Artifical Intelligence*, 32:333–377, 1987. also MIT AI Technical Report 802 November, 1985.

[6] Richard E. Fikes, Paul E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972. Also in Readings in Artificial Intelligence, 1980.

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

[8] R. James Firby. An investigation into reactive planning in complex domains. In *AAAI*, pages 202–206, 1987.

[9] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *AAAI*, pages 677–682, 1987.

[10] Patrick Hayes. The frame problem and related problems in artificial intelligence. In A. Elithorn and D. Jones, editors, *Artificial and Human Thinking*, pages 45–49. San Francisco: Jossey-Bass, 1973. Also in Readings In Artificial Intelligence, 1980.

[11] John H. Holland, Keith F. Holyoak, Richard E. Nisbett, and Paul R. Thagard. *Induction: processes of inference, learning, and discovery.* MIT Press, 1986.

[12] L. P. Kaelbling. Goals as parallel program specifications. In *Proceedings of AAAI-88*, page 60, 1988.

[13] Nils J. Nilsson. Shakey the robot. Technical Note 323, SRI AI Center, 1984.

[14] Panel on Planning and Execution. Rochester planning workshop, 1988.

[15] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[16] E. D. Sacerdoti. *A Structure for Plans and Behavior.* New York: Elsevier, 1977.

[17] M. J. Schoppers. Universal plans for reactive robots in unpredictable domains. In *Proceedings of IJCAI-87*, 1987.

[18] Austin Tate. Interacting goals and their use. In *Advance Papers of the 4th IJCAI*, 1975.

[19] D. E. Wilkins. Domain independent planning: representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.

[20] D. E. Wilkins. Recovering from execution errors in SIPE. *Computational Intelligence*, 1:33–45, 1985.