REPORT DOCUMENTATION PAGE

AD-A255 709

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 4, 1992	Final Report 1 Dec 90 - 31 May 92

4. TITLE AND SUBTITLE
A Probabilistic Approach to Anytime Algorithm for Intelligent Real-Time Problem Solving (U)

5. FUNDING NUMBERS
61102F
2304/A7

6. AUTHOR(S)
Josh Tenenberg and James Allen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Department
University of Rochester
Rochester, NY 14627-0226

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 92 0708

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
Building 410
Bolling AFB DC 20332-6448

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFOSR-91-0108

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved For Public Release;
Distribution of this document is unlimited.

12b. DISTRIBUTION CODE
UL

13. ABSTRACT (Maximum 200 words)

Our work on real time intelligent problem solving has focussed on the tradeoff between deliberation and activity. Such a tradeoff is required, since an excess of deliberation will be defeated by the dynamical nature of the world and by errors in the predictive model, and a lack of deliberation will not provide the agent with sufficient flexibility to perform well in novel situations. Our framework for evaluating this tradeoff includes both an explicit and an implicit component. In the explicit work, we represent the uncertainties associated with inaccuracies in the model and the inability to completely monitor changes in the world by expanding our language to include probabilities, and making choices about *when* to act and when to deliberate further based upon these explicit uncertainty measures. In the implicit approach, we use reinforcement learning of a Markov Decision Process to place a strict bound on deliberation. The agent's knowledge is obtained through an active sensory system having limited bandwidth, overcoming the standard limitations of assuming complete knowledge, but requiring modifications to the standard learning algorithm. Learning time is decreased by the use of social learning mechanisms as well as task decomposition and dynamic policy merging.

14. SUBJECT TERMS
Planning, reinforcement learning, probabilistic planning

15. NUMBER OF PAGES
36

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Final Report: A Probabilistic Approach to Anytime Algorithms for Intelligent Real-Time Problem Solving

Investigators: Josh Tenenberg and James Allen

August, 1992

Abstract

Our work on real time intelligent problem solving has focussed on the tradeoff between deliberation and activity. Such a tradeoff is required, since an excess of deliberation will be defeated by the dynamical nature of the world and by errors in the predictive model, and a lack of deliberation will not provide the agent with sufficient flexibility to perform well in novel situations. Our framework for evaluating this tradeoff includes both an explicit and an implicit component. In the explicit work, we represent the uncertainties associated with inaccuracies in the model and the inability to completely monitor changes in the world by expanding our language to include probabilities, and making choices about *when* to act and when to deliberate further based upon these explicit uncertainty measures. In the implicit approach, we use reinforcement learning of a Markov Decision Process to place a strict bound on deliberation. The agent's knowledge is obtained through an active sensory system having limited bandwidth, overcoming the standard limitations of assuming complete knowledge, but requiring modifications to the standard learning algorithm. Learning time is decreased by the use of social learning mechanisms as well as task decomposition and dynamic policy merging.

1 Overview

We have been investigating the tradeoff between deliberation and action in a real time planning agent. Traditional planning efforts in artificial intelligence [Fikes and Nilsson, 1971; Wilkins, 1984; Sacerdoti, 1974; Chapman, 1987; Allen, 1983] have been based on the agent's activity being purely mental, under the assumption that mental search costs less than physical search in the world. Problem solving amounts to having a model of both world state and the effects of the agent's actions, and using this model to generate hypothetical future states. If a future mental state is found that satisfies the goal, the agent simply "records" and then "runs" the sequence of actions that produced this hypothetical state. Note that this model will include not only predictive "rules, " or causal and relational axioms (e.g. "whatever the agent carries will also be At the destination the agent moves to"), but also concrete "facts" about particular relations or events at particular times (e.g., "the report is now in the brown briefcase"). This embodies the idea that it is better for an agent's "ideas to die in their stead", to paraphrase Popper.

But this traditional view presupposes an accurate, complete world model. Without such an accurate model, an agent's predictions will degrade (with such degradation compounding) as they project further in space/time. That is, predictions will increasingly become incorrect. However, in many domains, having access to a complete/correct predictive model will be infeasible or impossible, especially given the absence of expert knowledge by the system designer, the inability to formalize such expert knowledge even if it is available, limited sensory capabilities of the automated agent, and the unknown or unpredictable activities of other independent agents. Intelligent agents will thus be required to deal with an incomplete model.

Adding reactive modules at the lower level, as in [Chapman, 1987; Firby, 1987], aims toward addressing this problem by trading off deliberation for action and reaction. Implicit in the reactive approach has been the notion that at some point the relative cost of further planning exceeds its expected benefit. There are several issues, however, that require addressing, which most previous attempts at combining deliberation and action have failed to address. For instance, how will the reactive modules be encoded? How adaptable will they be to new information? Hard-coding such modules is costly, fragile to changes in design specification, and error-prone. In addition, how does the executive, deliberative reasoner choose between the different modules, and between further deliberation versus relinquishing control to a reactive module?

We have pursued both an explicit and an implicit approach to tackle different subsets of these issues. These two approaches will be briefly summarized here, and described in more detail below. One approach involves explicitly representing uncertainties by expanding ones language to include probabilities, and making choices about *when* to act and when to deliberate further based upon these explicit uncertainty measures. Although there have been several attempts at incor-

porating probabilities into planning [Kanazawa and Dean, 1989; Haddawy, 1990; Hanks, 1988], ours is the first at developing a language for combining an expressive temporal interval reasoner [Allen, 1984] with probabilities derived explicitly from a statistical analysis of the planner's own problem solving activitiesKyburg74. This makes important contributions for planners that must specify plans at the appropriate level of detail for an intelligent reactive module, and in approaching the well-known *qualification problem* [McCarthy, 1977]. Thus, the deliberator is making choices about when to act, and is monitoring the success or failure of its choices to condition future activity.

In the implicit approach, only predictions a single "step" into the future are made, in order to guarantee a constant time action cycle. However, by coupling reinforcement learning techniques with recognition of the current state through sensory input, the agent is able to incrementally construct an optimal action policy to guide its decisions through the state space. Thus, in some senses, a predictive model is implicitly encoded, but through action and adaptation over time, rather than mental search over a model. In this regard, we have made a number of important scientific contributions in learning and using optimal decision policies for real-time control modeled as Markov Decision Processes (MDP's). First, we show how the use of *indexicals* can be combined with an MDP representation in order to incorporate an active sensory/motor system. This is significant, in that previous efforts have assumed that the agent has direct access (not through the sensory system) to complete world state descriptions. In this way, we demonstrate that reinforcement learning techniques can be used to *control* perpetual focus; conversely, perceptual control allows significant scale-up in the size of problems that can be solved. Another important advance has been in the use of social learning mechanisms – observing other agents, and getting expert criticism – both of which afford dramatic improvements in learning optimal action policies. We are additionally exploring how the agent can scale the policies it learns to novel, complex problems, by incorporating a modular control design. This control design simplifies the learning task considerably, and also allows for the agent to integrate reactive and deliberative activity

In the following section, we describe the explicit approach that integrates statistics into a temporal interval reasoner to provide decision theoretic planning tools to a real time planner. We then discuss our implicit approach that extends previous work on reinforcement learning. We begin with a discussion of the Markov Decision Process formalism, followed by a brief discussion of unsupervised reinforcement learning. We then present our work on learning and sensing, the perceptual aliasing problem and the use of a perceptual sub-cycle to solve the problems that arise. We then discuss scaling the approach to more complex problems through task decomposition and social learning mechanisms. In Section 7, we summarize the research activities of the researchers involved in this contract, and list the contract related publications.

2 Planning with Statistics

There are two aspects of real time planning: generating plans in real time and generating real time plans. The first is characterized by limitations on the amount of time available when the plan is generated. For example, dealing with cars as one drives down the highway involves real time planning. One needs to react quickly to a dynamic and changing environment. We call this planning in real time. The second aspect of real time planning is the development of plans to deal with real time situations. Such planning is characterized by constraints on the availability of time when the plan is executed. For example, planning the sequences of turns that one will need to make as one drives down the road and the landmarks to look for involves this type of real time planning. One still needs to deal with a dynamic and changing environment, but one has time to reason about the problem before facing it. We call this generating real time plans.

Probability and decision theory are useful to both of these aspects of real time planning. The difference between these two types of real time planning involve a difference in a temporally indexed utility function. Planning in real time involves a utility function that decays with the passing of time. Here the planner realizes ever less utility as time passes so it gains a benefit from reasoning quickly. The longer it reasons the less utility it gains from its actions. Of course, the probability of achieving a high utility action may increase with planning, but this represents the tradeoff the planner faces. Generating real time plans require a more complex utility function. Here the temporal aspect of the utility function need not be monotonically decreasing. Instead, the utility of various actions depends on the time when the actions are taken. The planner tries to maximize expected utility as usual, but the temporal index on the utility function determines when the actions whould be taken.

Decision theory relies heavily on probability theory. In areas like medicine where articulate experts have access to statistical data, probability can represent these experts' uncertainty. Unfortunately, many planning domains lack such articulate experts, making this approach impractical. This is particularly true in domains involving real time response as evidenced by the difficulty of programming such systems. As an alternative when experts are unavailable, probabilities can be estimated from observations. For example, though no one knows the probability distribution on the length of time customers leave things under tables in fast food restaurants, robots designed to clean such restaurants might benefit from this information. The robot cleaning the floor would, however, have the opportunity to gather information about the likelihood of things being left on the floor. From these observations, it could estimate the desired probability. Moreover, this robot can use information about the precision of the estimates it has computed.

Three assertions summarize the difference between this work and previous work on using decision theory to deal with real time planning.

1. Probability is compiled experience.

2. The strength of the evidence on which these probabilities are based can be maintained with the probabilities at little extra cost.

3. An agent can control its inference using the strength of the evidence that gives rise to the probability.

Planners choose sets of actions relative to hypothetical futures to achieve goals. To do this, a planner must represent its knowledge of hypothetical futures, and reason with that knowledge. Because hypothetical futures are derived from a model, planners can have perfect information about them. Unfortunately, they rarely have complete information about *real* future world states. Almost invariably when a planner hypothesizes that an action causes event, there is a possibility that executing the real action corresponding to the hypothetical action may cause a real event that does not correspond to the hypothesized result. Planners are successful when the possibility of mismatch between their beliefs and the real world is unlikely. Most planners, both human and computer, do not reason about likelihood when generating plans and must therefore ignore unlikely occurrences. For example, one expects a car to start when one turns the key. If, however, the condenser has burnt out, the car will not start. One does not consider the possibility that the condenser has burnt out when planning to go to the store. Though the plan might fail, people can usually deal with unexpected contingencies in the execution of their plans. Problems arise because naively implemented logical theories are both too strong and too weak to represent the beliefs a planner needs to reason about the effects of its actions. Such theories are too weak because the complexity of the real world makes encoding it in a set of sentences difficult. The problem with strength appears in the difficulty of inferring properties of the world from beliefs.

The difficulty of generating useful theories of planning in intuitive languages have lead some to reject logic as the basis of an effective knowledge representation scheme. Still, the clear distinction logic makes between syntax and semantics, and the ability to prove soundness relative to a semantics gives it an advantage over all other knowledge representation schemes. One wants a logical language that deals with realistic problems effectively and probability gives a means of representing many aspects of the world that would otherwise be difficult in a logical language. Axiomatizing probability as a function in a first order language overcomes many of the problems of using logic to represent the knowledge a planner needs. Moreover, logic can ground probability. It can give one the ability to prove the soundness of statements concerning probability relative to a set of models.

Probability provides a useful numerical representation of uncertainty, but, as it is most commonly used in planning, the probability of all of the effects of the plans must be known before hand. In many planning domains, however, such experts may not exist; the planner itself may be the best expert in its field. We have developed a representation of probability that allows planners to reason about the relationship between probability and their experience.

If a planner must calculate its probabilities, it must decide when it has enough information to be confident of its calculations. Deciding when one is sufficiently confident of probabilities generated from observations is called the *sample size problem.* The sample size problem will be ubiquitous for planners that calculate probabilities from their experience. The most immediate incarnation of this problem is that of deciding whether choices are warranted by the evidence. A planner should make its decisions based on the probabilities about which it has good evidence, and discount the probabilities about which it is uncertain.

In addition to their inability to represent uncertainty, traditional planners have difficulty reasoning about concurrent actions, effects they themselves have not caused, and abstraction. Allen, Kautz, Pelevin and Tenenberg [Allen *et al.*, 1991] have developed a language for planning using temporal reasoning. This language deals with concurrent actions, but it cannot deal with partial interactions between actions. That is, though one say that two people have to lift simultaneously to pick up a piano, one cannot say that three people are more likely to be able to lift a piano than are two people. The knowledge representation language developed here extends Allen's with the ability to represent statistical probabilities. Four principles guide the extension of this temporally explicit planning language:

1. developing a planning system *should not* require the programmer to provide estimates of real valued parameters,

2. the planning system should perform reasonably even when it has no experience,

3. the planning system should be able to recognize and deal with uncertainty,

4. the planning system built using this system should be able to use its experience.

Temporal reasoning provides a means of dealing with many of the problems involved in planning [McDermott, 1982; Allen, 1983; Allen, 1984; Dean, 1985; Shoham, 1988; Koomen, 1989; Allen, 1991]. Planners can reason about the passing of time and concurrent execution using temporal logic, but temporal reasoning cannot deal with incomplete information. Time provides a structure into which a planner can fit its plans giving a planner the ability to generate plans by reasoning about the relative temporal location of the actions in hypothetical futures it tries to bring about. However, time does not provide the structure for deciding which hypothetical futures are more likely.

Events are central to the logic described here because, instead of providing probabilities, the programmer describes the events the planner is likely to encounter. From statistics on encounters with these events it computes constraints on probabilities. Events underlie the intuitions behind Allen's interval temporal logic [Allen, 1983; Allen, 1984], and provide a structure within which statistics can be collected.

To collect statistics about the occurrences of events, a distinction between event instances and event types must be made: event instances might actually occur; event

types are sets of those instances. This distinction allows the planner to deal both with reasoning about likely futures, and with reasoning about a particular future. The planner reasons about the likelihood of events by reasoning about similar event types it has observed in the past; it reasons about interactions between action events by reasoning about event instances of this type. That is, it reasons about the instances of events it will try to bring about and about the types of events it has seen in the past. For example, the planner may reason that, since the collection of event instances in which the ignition key was turned covers the collection of event instances in which the car started, turning the key is a useful action in starting a car. It may then reason that to start the car at noon, it must have the key in its hand at noon. Probability is defined to be the ratio of one type of events to another, and the planner keeps track of the number of events of each type it sees. It then constrains probabilities based on its evidence so far.

The formalism also allows a programmer to represent probabilities if they are available, or to express incomplete knowledge of them if only limited information is available. The planner generates a plan of actions that, based on its previous experiences, it believes will bring about events that are likely to lead to the goal. Because the language is an extension of the language presented by Allen [Allen *et al.*, 1991], the programmer can encode the knowledge necessary for checking the consistency of the plans.

The conditional probability of one event type given another is defined to be the ratio of the number of event instances in those event types, so the planner can reason about probabilities based on its experience. This treatment of probability, which is often called frequentist, starts with a definition of a random experiment [Bickel and Doksum, 1977]. A random experiment consists of observing the outcome of a set of circumstances.

Unfortunately, probabilities defined as ratios of event types are usually impossible to calculate. The only event types about which one can know the probability are those whose constituent event instances have already occurred, whereas actions must be chosen for events that have not yet occurred. Fortunately, partial knowledge of these ratios constrains them. For example, if a coin lands heads five times, one cannot determine with complete certainty the probability of that coin landing heads. However, one can state with complete certainty that the probability of a coin landing heads cannot be 0. Confidence intervals [Neyman, 1960] are a standard way of dealing with such partial information, and will be used throughout the thesis.

The planner uses the language described above to encode knowledge that determines the probability function used in its decision theory. This decision theory has two novel features. First, because the language uses interval estimates of probabilities, interval valued expected utilities result. This decision theory generates a partial order over possible plans rather than a total order. When this decision theory fails to distinguish plans, the planner must choose among the plans by other means. Second, the planner can always estimate the value of a needed probability. In the worst case,

when no probabilities have been asserted and it has no experience, the planner reasons that the probability lies in the interval [0,1]. Though ,interval valued expected utilities makes choosing between possible actions difficult, it also give the agent access to another type of knowledge. The width of the expected utility interval encodes the depth of the agents experience with a choice.

Humans also face such inabilities to choose, and these inabilities frequently result from insufficient experience with the actions. For example, a novice faced with the choice of checking either the battery or the carburetor for an explanation of a car that does not start may be unable to make this choice. A mechanic, on the other hand, may know that batteries are much more likely to go bad than are carburetors and would check the battery first. The mechanic may be wrong, but he or she faces no indecision.

The novice will make a choice, and will check either the battery or the carburetor first. Making choices in the face of insufficient information is a necessary task for any agent that bases its decisions on probabilities gained from experience. There are several ways in which such an agent might make such a choice.

1. The agent might make a choice with lower confidence. Ultimately, it might guess by making a decision based on a maximum likelihood estimate.

2. The agent might make a choice by abstracting the situation. For example, it might decide that in its experience, electrical systems are more unreliable than hydraulic system, and check the battery first.

3. The agent might gather more information that might disambiguate the system. For example, an agent might call someone else to ask which to check first.

4. The agent might forgo the decision, and make a decision between an altogether different set of actions.

Much of our work with statistically based planning is elaborating the possible strategies of an agent faced with the problem of making a choice with insufficient information.

Developing a planning system that uses decision theory to generate plans will fail because it is unlikely to have sufficiently detailed knowledge of the requisite probabilities, and, if it does, it is unlikely to be able to perform all of the computations necessary to generate the plan in a reasonable amount of time. However, the planner can provide the following services on which a traditional planner might rely.

1. Calculation of the most likely action to achieve a goal in a situation.

2. Executing plans after they are generated.

3. Calculating the aspects of a situation that may effect the outcome of an action

Our efforts have concentrated on the first two services.

To request a choice of action, the plan generation module presents the decision theory module with a desired effect. The decision theory module then chooses the set of actions with the highest probabilities, according the the modules best estimate. The module then chooses the most specific of these actions.

To do this, the module must be given: a confidence level, a list of candidate actions, an event type consisting of events similar to those occurring when the action is to be taken, and a desired effect of the action. The program then calculates a confidence interval at the level specified for the probability that the effect event type will occur given that the action is taken in the event type specified. The set of actions with the highest probability is returned. That is, it returns the set of actions for which no other action has a confidence interval whose lower bound is greater than its upper bound. This set represent the set of actions with the best probability of success given the observations the planner has made to date.

The decision theory module can execute plans for the planner. In doing so it uses probabilities and utilities to provide three services. First, it monitors the execution of the plan. It must do this anyway because it has to know the effects of its actions to maintain statistics on them. It uses its knowledge of the strength of its information to project the effects of actions. If it has strong statistical evidence that a sequence of actions will achieve the desired effects, it can execute that sequence. If it does not, it executes the sequence for which it has sufficient information, and checks the state of the world at the end of the actions it does execute. In this it exhibits human like behavior. In well practiced routines it executes complex programs without monitoring them. For routines it has practiced less well, it continually checks to see that the conditions it needs in order to continue are met.

On the theoretical side, the research has resulted in a first order language of time, probability and statistics that provides a grounding for robot planning languages that reason about uncertainty from the robot's experience. In addition, it describes constraints on the utility functions useful to planners based on a set of prioritized goals present to the planner by the user. Finally, it describes a decision theory based on the probabilities generated from the planner's experience and the utility function based on the users goals.

The formal first order language is developed in two stages: first a language of time, events and probability is presented; then the language is extended to reason about situations of incomplete knowledge. The language of time, events and probabilities is shown to be sound relative to appropriate models. This language differs from Haddawy's [Haddawy, 1991] and Kanazawa's [Kanazawa, 1991] in that it allows one to tailor the range of the probability function to the problem at hand. The relationship between time and probability is defined only for event types (ie. sets of event instances) not for arbitrary sentences in the language. One could define an event type for every sentence in the language, and the language would them be as expressive as Haddawy's and Kanazawa's. By defining probability only over event types, however,

a programmer developing a system using the language can specify the aspects of the world over which statistics are maintained. That is, instead of reasoning about the σ-field defined by the fundamental probability set generated by all possible predicates in the language, one can reason about a σ-field defined from a set of subsets of the fundamental probability set. The language allows one to specify the particular subsets of the fundamental probability set that define the σ-field over which, in turn, define the range of the system's probability function. This capability is unique, and promises to be useful. Defining probability over only part of the language reduces the burden of specifying the probability function.

Another interesting result relative to the formal first order language is that theories in the language remain sound for the same set of models even when the theories are abridged in a certain way. The probability of event types is defined only when the system knows the cardinality of those event types. Even when the system's knowledge of the cardinality of event types is restricted, however, it can still make useful inferences about probability. These inferences are sound relative to all models of a theory containing complete information about the cardinality of the event types. Information about the cardinality of the models is restricted relative to temporal intervals, so the system can reason about the strongest constraints it can place on the probability of an event type relative to an interval. Planners can apply this capability to reason about the strongest constraints on probabilities given its experience to date.

On the applied side, the research has resulted in the implementation of a decision theoretic planning assistant. The first order language developed theoretically grounds the planning language in which a programmer specifies the module. Two examples of such planning modules have been built and have been tested experimentally. The experiments show how one can use the knowledge representation language presented here to develop a planning module that aids a robot's planning system by reasoning about the robot's experience.

Traditional planners, which reason about actions whose effects are certain, could extend their effectiveness by using the decision theoretic planning assistant. The planner insures that the choices the decision theoretic assistant reasons about do not interfere with each other. Because the planning assistant knows that the actions do not interfere with each other, it does not need to consider the effects of previous actions in the situation for which it is making a choice. In return, the decision theoretic planning module performs two services for the traditional planner. It aids the planner by making choices among a set of alternatives presented by the planner. The theoretical work mentioned above justifies these choices. In addition, the planning module executes plans by attempting, as far as possible, to discharge the assumptions of certainty made by the planner. The module aids the planner reasoning in the TRAINS domain, a multi-agent domain involving manufacturing and transportation. The planning module should provide guidance to those interested in developing a similar module for dynamic, uncertain, multi-agent domains.

The decision theoretic module also gathers the information that allows it to make

better choices on subsequent queries. It bases its decisions on constraints on probability inferred from observations, from probabilities asserted by the programmer, and from facts inferred from its knowledge base. The module gathers information by observing events and inferring from its observations the type of events that must have occurred to cause such an observation. It then updates the statistics on the events observed and the events inferred. Again, the design of this module should aid in the design of modules for gathering information applicable to planning.

3 Learning, Acting, and Sensing

3.1 Markov Decision Processes

The above research assumes that there exist intelligent, reactive modules for performing sensory/motor interactions with its environment, over which the explicit monitoring and probability reasoning are performed. In this next section, research is described for building such modules.

We will assume that the robot-environment interaction can be modeled as a Markov decision process. In a Markov decision process (MDP), the robot and the environment are modeled by two synchronized finite state automatons interacting in a discrete time cyclical process. At each point in time, the following series of events occur.

1. The robot senses the current state of the environment.
2. Based on the current state, the robot chooses an action to execute and communicates it to the environment.
3. Based on the action issued by the robot and its current state, the environment makes a transition to a new state and generates a reward.
4. The reward is passed back to the robot.

Let S denote the set of possible environmental states, and let A denote the set of possible actions. We will assume that both S and A are discrete and finite.

The dynamics of state transitions are modeled by a *transition function*, T, which maps state-action pairs into next states ($T : S \times A \rightarrow S$). In general, the transition function may be probabilistic. If we let x_t denote the state at time t, let a_t denote the action selected at t, and let X_{t+1} be the random variable denoting the state at time $t+1$, then $X_{t+1} = T(x_t, a_t)$. T is usually specified in terms of a set of *transition probabilities*, $P_{x,y}(a)$, where

$$P_{x,y}(a) = Prob(T(x,a) = y). \tag{1}$$

Rewards generated by the environment are determined by a *reward function*, R, which maps state into scalar rewards ($R : S \rightarrow \Re$). In general, the reward function

may also be probabilistic. If we let R_t be the random variable denoting the reward received at time t, then $R_t = R(x_t)$. For simplicity, we shall hereafter assume that rewards are deterministic, and depend only upon the current state. Notice that in a MDP, the effects of actions (in terms of the next state and immediate reward received) only depend upon the current state. Process models of this type are said to be memoryless and satisfy the *Markov Property.*

The robot's objective is to learn a control policy that maximizes some measure of the total reward accumulated over time. In principle, any number of reward measures can be used, however, the most prevalent measure is one based on a discounted sum of the reward received over time. This sum is called the *return* and is defined for time t as

$$return(t) = \sum_{n=0}^{\infty} \gamma^n r_{t+n} \tag{2}$$

where γ, called the temporal discount factor, is a constant between 0 and 1, and r_{t+n} is the reward received at time $t+n$. Because the process may be stochastic, the robot's objective is to find a policy, f^*, the maximizes the *expected return.*

An important property of MDPs is that f^* is well defined and guaranteed to exist. In particular, the *Optimality Theorem* from dynamic programming [Bellman, 1957] guarantees that for a discrete time, discrete state Markov decision problem there always exists a deterministic policy that is optimal. Furthermore, a policy f is optimal if and only if

$$Q_f(x, f(x)) = \max_{a \in A}(Q_f(x, a)) \qquad \forall x \in S \tag{3}$$

where $Q_f(x, a)$, called the *action-value* for state-action pair (x, a), is defined as the return the robot expects to receive given that it starts in state x, applies action a next, and then follows policy f thereafter [Bellman, 1957; Bertsekas, 1987]. Intuitively, Equation 3 says that a policy is optimal if and only if in each state, x, the policy specifies the action that maximizes x's action-value.

If an MDP is completely specified *a priori* (including the transition probabilities and reward distributions) then an optimal policy can be computed directly using techniques from dynamic programming [Bellman, 1957; Ross, 1983; Bertsekas, 1987]. Because we are interested in robot learning, we shall assume that only the state space S and set of possible actions A are known *a priori* and that the statistics governing T and R are *unknown.* Under these circumstances the robot cannot compute the optimal policy directly, but must explore its environment and learn an optimal policy by trial-and-error. Research in reinforcement learning has addressed this task.

3.2 Reinforcement Learning

In our work we have focussed on a single reinforcement learning algorithm called Q-learning [Watkins, 1989]. Other, similar reinforcement learning algorithms could have

$Q \leftarrow$ a set of initial values for the action-value function (e.g., uniformly zero)
For each $x \in S$: $f(x) \leftarrow a$ such that $Q(x,a) = \max_{b \in A} Q(x,b)$,
Repeat forever:

1) $x \leftarrow$ the current state
2) Select an action a to execute that is usually consistent with f but occasionally an alternate. For example, one might choose to follow f with probability p and choose a random action otherwise.
3) Execute action a, and let y be the next state and r be the reward received.
4) Update $Q(x,a)$, the action-value estimate for the state-action pair (x,a):
   $Q(x,a) \leftarrow (1-\alpha)Q(x,a) + \alpha[r + \gamma U(y)]$
   where $U(y) = Q(y, f(y))$.
5) Update the policy f:
   $f(x) \leftarrow a$ such that $Q(x,a) = \max_{b \in A} Q(x,b)$,

Figure 1: A simple version of the 1-step Q-learning algorithm.

been used as well with much the same results; however, we shall focus on just this one. For alternatives see for instance [Barto *et al.*, 1983; Holland, 1986; Williams, 1987; Schmidhuber, 1990].

In Q-learning the robot estimates the optimal action-value function directly, and then uses it to derive a control policy using the local greedy strategy. A simple version of a Q-learning algorithm is shown in Figure 1. The first step of the algorithm is to initialize the robot's action-value function, Q. Q is the robot's estimate of the optimal action-value function. If some prior knowledge about the task is available, that information may be encoded in the initial values, otherwise the initial values can be arbitrary (e.g., uniformly zero). Next the robot's initial control policy, f, is established. This is done by assigning to $f(x)$ the action that locally maximizes the action-value. That is,

$$f(x) \leftarrow a \quad \text{such that} \quad Q(x,a) = \max_{b \in A} Q(x,b), \tag{4}$$

where ties are broken arbitrarily. The robot then enters a cycle of acting and policy updating. First, the robot senses the current state, x. It then selects an action a to perform next. Most of the time, this action will be the action specified by the robot's policy $f(x)$, but occasionally the robot will choose a random action.[1] The robot executes the selected action and notes the immediate reward r and the resulting

[1]Occasionally choosing an action at random is a particularly simple mechanism for exploring the

state y. The action-value estimate for state action pair (x, a) is then updated. In particular, an estimate for $Q^*(x, a)$ is obtained by combining the immediate reward r with a utility estimate for the next state, $U(y) = \max_{b \in A}[Q(y, b)]$. The sum

$$r + \gamma U(y), \tag{5}$$

called a 1-step corrected estimator, is an unbiased estimator for $Q^*(x, a)$ when $Q = Q^*$, since, by definition

$$Q^*(x, a) = E[R(x, a) + \gamma V^*(T(x, a))], \tag{6}$$

where $V^*(x) = \max_{a \in A} Q^*(x, a)$. The 1-step estimate is combined with the old estimate for $Q(x, a)$ using a weighted sum:

$$Q(x, a) \leftarrow (1 - \alpha)Q(x, a) + \alpha[r + \gamma U(y)], \tag{7}$$

where α is the learning rate. Finally, the robot's control policy is updated, and the cycle repeats.

3.3 Completeness Assumptions and Indexicals

The action policies that are found as a result of the trial and error of the Q-learning algorithms can be viewed as compilations of past problem-solving experience, in that the agent learns, for each state, to select the action that will give the greatest long-run return. Unfortunately, these optimality guarantees rely upon assumptions that the agent has complete knowledge about all relevant facts in the world. Further, the time required to learn often scales exponentially as the size of the state space increases, even if the task itself has not changed (e.g., by adding additional extraneous assertions or predicates to the state descriptions).

By dropping the completeness assumption and supposing that there is a sensory system that mediates between the world and the agent, we have taken a large step in developing more realistic models of agency and behavior. On the one hand, this makes the control task more difficult, since the agent is only able to attend to a fraction of the world at one time. That is, at any moment, the agent's *representation* of the world is inherently incomplete, thus making it considerably more challenging to determine what the optimal action should be. On the other hand, by *not* attending to certain aspects of the world, the agent is able to abstract its representation, and thus gain efficiencies from this simpler representation. By using indexical reference to objects and having propositions in the language denote functional features of objects, rather than unique, arbitrary, names as in standard predictive planning systems, the agent is able to learn tasks even in the face of a considerable number of additional extraneous objects.

environment. Exploration is necessary to guarantee that the robot will eventually learn an optimal policy. For examples of more sophisticated exploration strategies see [Kaelbling, 1990; Thrun, 1992; Sutton, 1990].

Unfortunately, the use of indexicals and the sensory system to encode world state leads to errors in standard reinforcement learning techniques, due to what we have termed *perceptual aliasing.* Perceptual aliasing occurs because more than one external world state might have identical internal encodings as represented by the partial state information obtained from the sensors. Thus, a policy recommendation that is good for one of the aliased states may be poor for another. As a simple example, consider a blocks world agent that can attend to objects in the world, knows the colors of the objects that it is attending to (Blue, Red, or Green), and knows the height of the stack above the object it is attending to. Then, assuming that the agent is attending to the Blue block in Figure 2 (marked by a +), the two different world states have the

Figure 2: Aliased World States: Two states having the same internal representation.

same internal representation – in both cases, the agent is attending to a blue block that has no other blocks on top of it. This is problematic if the agent's optimal action with respect to a particular goal (such as, stacking the Blue block on the Green one) depends upon distinguishing these two different states. In this case, the agent does not stabilize on an optimal policy.

3.4 Dealing with Perceptual Aliasing

Perceptual aliasing can be a blessing or a curse. If the mapping between the external world and the internal representation is chosen correctly, a potentially huge state space (with all its irrelevant variation) collapses onto a small simple internal state space. Ideally, this projection will group world situations that are the same with respect to the task at hand. But, if the mapping is not chosen carefully, inconsistencies will arise and prevent the system from learning an adequate control strategy. The main result of our study is a first attempt at a decision system, based on reinforcement learning, that can cope with perceptual aliasing. The new decision system is designed specifically to be embedded within an agent with an active sensory-motor system and to actively control perception to overcome the negative effects of perceptual aliasing. The decision system learns not only the correct overt actions needed to solve a problem, but also how to control its sensory subsystem in order to focus on those objects in the world that are relevant to the task.

The design is based on three observations/assumptions:

1. In active perception a world state can be represented by multiple internal states, one of which is usually consistent. That is, in any given state, if the agent looks

around enough it will eventually attend to those objects that are relevant to the task, and the internal state associated with that sensory configuration will be consistent. Our algorithm depends on the existence of one consistent internal state for each world state.

2. Inconsistent states disrupt the decision system's ability to learn by promising erroneously large expected returns. If we can detect inconsistent states and actively lower their action-value estimates, we can minimize their negative effects.

3. If the world is deterministic, then inconsistent states will (because of averaging) periodically overestimate the utility of the actual world state, whereas the incidence of overestimation in consistent states can be made to diminish with time. Therefore, inconsistent states can be detected by monitoring the sign of the estimation error in the standard Q-learning updating rule.

3.5 The Overt Cycle

The algorithm used by the new decision system is described as follows. The decision subsystem recognizes two classes of internal action commands (A_I): overt actions and perceptual actions, denoted A_O and A_P, respectively. Overt actions change the state of the external world whereas perceptual actions change the mapping between world and the internal states.[2]

The main decision cycle is the overt cycle, which concerns itself with choosing overt actions in an attempt to maximize return. Embedded within the overt cycle is a perceptual cycle. After each overt action, the system executes a series of perceptual actions (the perceptual cycle) in an attempt to assess the true state of the external world. The objective of the perceptual cycle is to find an internal state that is a consistent representation of the current world state. Upon completion, the perceptual cycle returns a list, S_t, of the internal states encountered during the perceptual cycle. Each state corresponds to a different view (representation) of the current external world. The utility of the current world state, $V_E^*(x_t)$, is estimated as the maximum utility of the individual internal states, $max_{s \in S_t}(V_I(s))$. As will be described below, our algorithm for adjusting the utility estimates of internal states severely lowers the utility estimates of inconsistent states. Consequently, utility estimates for world states tend to be based on the utilities of consistent states and not biased by the apparitional maxima associated with inconsistent states.

Once $V_E^*(x_t)$ has been estimated, the action-value estimates Q_I, for the previous overt action are updated (as described below). The overt cycle then continues by selecting an overt action to execute. With probability p (e.g., $p = 0.9$), the system chooses the action consistent with its policy; the rest of the time it chooses an action at random. When following policy, the action is chosen by searching among active

[2]As a side effect, overt actions may also change the perceptual configuration, but perceptual actions are not allowed to affect the world state.

internal states, S_t, for the decision with the largest action-value. That is, after collecting S_t, the system has $|S_t| \times |A_O|$ decisions it must consider, one for each possible action in each possible representation of the current state. We denote this set by D_t. The system's policy is to choose the decision, $d_\pi = (s_\pi, a_\pi)$, called the *policy decision*, such that

$$Q_I(s_\pi, a_\pi) = \max_{(s,a) \in S_t \times A_O} [Q_I(s,a)]. \tag{8}$$

Once an overt action is chosen, it is executed and the overt cycle begins anew.

3.6 Learning a New Action-Value Function

Standard Q-learning algorithms estimate the action-value of a decision as the return the system expects to receive given that it makes that decision and follows its policy thereafter. However, for inconsistent decisions this definition leads to artificially high action-values (aberrational maxima). We have developed a modified learning algorithm that is based on Q-learning but incorporates a competitive component. This component tends to suppress the action-values of inconsistent decisions while allowing action-values for consistent decisions to take on their nominal values. Since action-values for policy decisions are now based on predictions from consistent decisions, they more accurately estimate the true values of the actual decisions.

The learning algorithm is based on identifying one decision among D_t that takes the "lion's share" of the responsibility (credit or blame) for the outcome of the next overt action. We identify this decision as the *Lion*. If a_L is the next overt action to be executed by the system, then the lion is defined as the maximal decision among D_t that is consistent with the action a_L. That is,

$$Lion = (s_L, a_L) \text{ such that } Q_I(s_L, a_L) = \max_{s \in S_t}(Q_I(s, a_L)). \tag{9}$$

When the system is following its policy, the lion is just the policy decision, $Lion = d_\pi$.

The idea underlying the use of a lion is that in every situation the lion should be a consistent decision; whenever it is not, the inconsistency in the decision should be detected and its action-value should be suppressed. That is, we would like the decision system to learn a new action-value function in which the action-values of consistent decisions take on their actual values, and the action-values of inconsistent decisions are zero:

$$Q_I^{ideal}(s,a) = \begin{cases} Q_I^*(s,a) & \text{if } (s,a) \text{ is consistent} \\ 0 & \text{otherwise .} \end{cases} \tag{10}$$

Inconsistent lions are detected and suppressed as follows. If at time t, the action-value of the lion, $Q_I(s_L, a_L)$, is greater than the estimated return obtained after one step, $r_t + \gamma V_I(s_{t+1})$, then the lion is suspected of being inconsistent and the action-value associated with it is suppressed (e.g., reset to 0.0). Actively reducing the action-values of lions that are suspected of being inconsistent gives other (possibly consistent)

decisions an opportunity to become lions. If the lion does not overestimate the return, it is updated using the standard 1-step Q-learning rule. To prevent inconsistent decisions from climbing back into contention, the estimates for non-lion decisions in D_t are updated at a much lower learning rate and only in proportion to the error in the lion's estimate. The observation that allows this algorithm to work is that inconsistent decisions will eventually (at one time or another) overestimate their action-values and, thus, will eventually be suppressed. On the other hand, it can be shown that a consistent lion is stable (i.e., it will not overestimate its action-value) if every state between the lion's state and the goal also has a consistent policy decision. Thus, inconsistent decisions are unstable with respect to lionhood while consistent decisions eventually become stable.

3.7 The Perceptual Subcycle

The objective of the perceptual cycle is to accumulate a set of internal representations of the external world, one of which has a consistent policy decision. This goal is achieved by executing a series of perceptual actions. In our current implementation, each perceptual cycle executes a fixed number ($n = 4$) of perceptual actions. This number has proven adequate for our experiments, but it is easy to imagine variable length perceptual cycles in which the cycle either terminates as soon as a consistent internal state is found or increases when inconsistent states are encountered.[3] The algorithm for selecting actions within the perceptual cycle is similar to the algorithm for choosing overt actions in the overt cycle. With probability p' (e.g., $p' = 0.9$), the system follows its policy, otherwise it selects at random. When following policy, the action selected is the perceptual action a_p such that $Q_I(s, a_p) = \max_{b \in A_P}(Q_I(s, b))$, where s is the system's current internal state. That is, the policy calls for perceptual actions that lead to internal states with maximal expected returns.

The rules for updating action-values for perceptual actions are those for standard 1-step Q-learning, within the **Perceptual Cycle** procedure. These updating rules lead to action-values that *average* the utilities of the states that result from executing a perceptual action. Since consistent states tend to have higher utilities than inconsistent states (whose action-values are suppressed), the effect is to choose perceptual actions that lead to consistent internal states.

The lion algorithm's response to variation in the return estimates is extreme. The utility of a state that exhibits the slightest variation in return is suppressed. A similar, but somewhat less extreme approach has been used by Grefenstette in a classifier system called SAMUEL (Grefenstette, 1988; Grefenstette, 1989; Grefenstette *et al.*, 1990). In SAMUEL, the variance in a rule's return is estimated as well as its mean, and the strength of the rule is determined based on the difference between estimates

[3]Actually, it may be possible to eliminate the distinction between the overt cycle and the perceptual cycle and integrate them into a single cycle in which the action (either overt or perceptual) with the highest utility is chosen. We are currently experimenting with such an algorithm.

for its mean and its variance. Using this *variance-subtraction* formula, consistent rules tend to be favored over inconsistent ones whose returns vary.

While Grefenstette had success using variance-subtraction in a simple missile evasion task, it performed poorly when we applied it to a simple block stacking task. Our initial attempt to deal with perceptual aliasing in the block stacking task was to use exactly variance-subtraction. Only later did we develop the lion algorithm. There are at least two reasons why variance subtraction did not work for us. First, it is difficult to obtain accurate, unbiased estimates of the return variance since the world states associated with a given internal state are not encountered equally often. This is especially true one the system begins to converge on a policy. As a result, even a decision that is wildly inconsistent may have a small variance estimate and may not be suppressed. Also, variance subtraction does not guarantee that consistent decisions will eventually dominate — subtracting (even an accurate) variance estimate from the mean may not reduce the action-value of a decision enough to permit a competitor to dominate. As a result, inconsistent decisions may continue to participate in action-value/utility estimation and create aberrational maxima.

The particular circumstances that allow variance-subtraction to succeed in the missile evasion task studied by Grefenstette are difficult to obtain from the available literature (Grefenstette, 1988; Grefenstette, 1989; Grefenstette *et al.*, 1990; Ramsey *et al., 1990*). However, this issue is certainly worthy of further investigation as are other algorithms for detecting and coping with inconsistent decisions.

We believe this work to be a significant advance in machine learning and intelligent control, in that it drops the assumption that the world is completely represented, and uses an active sensory system to compensate for the perceptual aliasing that this give rise to. We are further investigating relaxing some of the assumptions upon which this approach rests, and expanding the scale of problems that it can solve. In particular, a naive learner with no *a priori* policy information may take an exponential amount of time to learn to solve particular problems. Further, much of this time is invested in early exploration of the state space when the agent has little experience to guide its behavior. In order to attack this problem, we have explored partitioning the representation of the state space, so that only those parts of the representation related to specific tasks will be used in learning those tasks. Such partitioning can be used to structure the representation hierarchically, and has been applied to the problem of visual object recognition. We have also used the modular approach in combination with a *predictive model* in order to develop policies for solving novel, conjunctive-goal tasks. Finally, we have pursued two *social* mechanisms for incorporating knowledge from other agents in the environment, termed learning with an external critic, and learning by watching, to speed up learning. These approaches are detailed in the following sections.

4 Multiple Goal Tasks

Intelligent behavior must involve the coordination of multiple activities. We have been exploring the control for autonomous robots that coordinate behavior in order to accomplish a series of tasks that change over time. For example, we can imagine an animal having such goals as getting food, getting water, procreating, tending to young, avoiding predators, and so on. Each goal will arise independently of the others, repeatedly over time, and in arbitrary combinations. Accordingly, the robot's behavior should be influenced by the set of goals that are currently active.

Instead of having just a single goal, suppose the robot has n goals, $\Gamma^1, \Gamma^2, \ldots, \Gamma^n$. Next, instead of having each goal be satisfied by a single state, associate with each goal Γ^i a set of satisfying states $G^i \subset S$. Finally, associate with each goal Γ^i a reward function R^i such that

$$R^i(x) = \begin{cases} c^i & \text{if } x \in G^i \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

where c^i is a scalar constant reward associated with each goal Γ^i.

This extension only goes part way towards our desired objective. At this point we have multiple goals; but the goals are static. Intelligent behavior is more dynamic and cyclic. Drives, purposes, and goals come and go with time, as the robot consumes and renews resources, as circadian cycles repeat, as opportunities and interruptions come and go. To obtain time varying goals we need to introduce the notion of goal activations. That is, we assume that each goal has associated with it a time dependent activation, which we denoted by ${g^i}_t$. For simplicity we assume that activations are binary valued processes, where ${g^i}_t = 1$ indicates that, at time t, Γ^i is active, and ${g^i}_t = 0$ indicates Γ^i is inactive. At each time point, then, we view each g^i as a bit, and the vector of all such bits we call the activation vector:

$$\bar{g} = g^1 \cdot g^2 \cdot g^3 \cdots g^n. \tag{12}$$

Intuitively, a goal's activation encodes its current importance. Functionally, activation values are used to modulate the reward received by the robot. In particular, we assume that the robot only receives a reward upon entering a goal state for an active goal. Under this new scheme, the reward function depends on both the current world state and the current activation vector:

$$R(x, \bar{g}) = \sum_{i=1}^{n} R^i(x) g^i. \tag{13}$$

We assume that the dynamics of goal activations are described by the following transition rules:
If ${g^i}_t = 1$, then

$${g^i}_{t+1} = \begin{cases} 0 & \text{if } x_t \in G^i \\ 1 & \text{otherwise.} \end{cases} \tag{14}$$

If $g^i_t = 0$, then

$$g^i_{t+1} = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1-p. \end{cases} \tag{15}$$

Using these transition rules, a goal, once activated, persists until it is achieved. Inactive goals on the other hand, spontaneously become active according to a Bernoulli process.

Multiple goal tasks of this form seem to more closely model the dynamics of tasks facing autonomous systems since different goals come and go over time and as the robot attends to its various needs as they arise.[4] Notice that in a multiple goal task there is no need to artificially partition the robot's behavior into trials; nor is it necessary to introduce operations that magically teleport the robot to a new point in the state space once a goal is reached. In a multiple goal task, when the robot accomplishes one goal, that source of reward disappears (at least temporarily) and the robot naturally moves on to perform other activities (and seek alternative sources of reward).

Multiple goal tasks can be modeled as Markov decision processes by extending the state space to include both the state of the external world and state of the activation vector. Formally, a multiple goal task is defined by the MDP (S_m, A_m, T_m, R_m), where

S_m is a composite state space that encodes both external world state and goal activations. $S_m = S \times \bar{G}$, where S is the set of possible world states and $\bar{G}$ is the set of possible goal activation vectors. For an n-goal task, $\bar{G} = \{0,1\}^n$.

A_m is the set of actions available to the robot.

T_m is the composite transition function on $S \times \bar{G} \times A_m$ into $S \times \bar{G}$. Specifically,

$$(x_{t+1}, \bar{g}_{t+1}) = T_m(x_t, \bar{g}_t, a_t) \tag{16}$$

where

$$x_{t+1} = T(x_t, a_t), \tag{17}$$

$$\bar{g}_{t+1} = T_A(\bar{g}_t, x_t), \tag{18}$$

T is the underlying transition function for external world states, and T_A is the transition function for goal activations which encodes the rules given above.

R_m is the composite reward function given by Equation 13.

[4] It is certainly possible to extend our definitions further, by introducing continuous valued goal activations and activation rates that are goal specific. However, our simple model is sufficient for the purposes of the current discussion.

We shall use f_m^* and Q_m^*, respectively, to denote the optimal policy and optimal action-value function for a given multi-goal task.

To illustrate, consider the example shown in Figure 3, which shows a The robot is

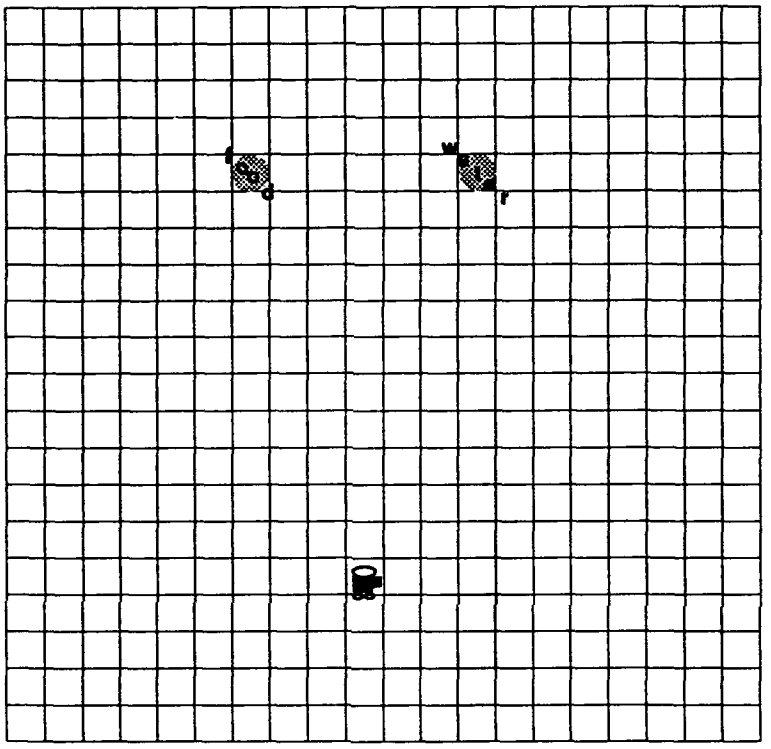

Figure 3: Example multi-goal problem

in the bottom part of the scene, and the food and water are the goal objects. The set of actions are the movements into adjacent locations. The states, however, encode not only position, but also the set of goals that are currently active (i.e., whether the robot needs to get food, water, or both). The robot's need for a particular object will arise according to a Bernoulli process, and once the object is obtained, the robot receives a reward, and the associated goal activation is set to 0. The robot then turns its efforts toward achieving its remaining active goals.

4.1 Learning Multiple Goal Tasks

Because multiple goal tasks, (as we have defined them), are MDPs, we know that Q-learning can be used to construct robots that are guaranteed to eventually learn an optimal control policy. However, the straightforward application of Q-learning, in a *monolithic* controller, leads to prohibitively slow learning. In particular, scaling from a single goal task to a n-goal task leads to an exponential growth in the state space.

Instead of having to learn action-values for $|S| \cdot |A|$ state-action pairs, $|S| \cdot 2^n \cdot |A|$ action-values must be learned. In general, if goal activations are allowed to take on a range of values, say m (where value encodes priority) the size of the state space scales as m^n. Without any means of generalizing (or interpolating) action-value estimates across state-action pairs, the learning time can be expected to scale exponentially in the number of goals.[5]

4.2 A Decomposition approach

The trouble with the monolithic approach is that experience gained in solving a goal under one set of circumstances is not easily transferred to solving that same goal in other similar circumstances. For example, in the task shown in Figure 3, the action-values updated when the active goals are `get food` and `get water` are distinct from the action-values updated when just the goal `get food` is active. Consider the case where the robot is very close to the water, and has only the goal of getting food. The robot should ignore its proximity to water, and go directly toward the food. However, if the robot has the goal of getting both food and water, from this same location, it should first obtain water before going toward the food. In general, the action-values for situations with identical world states but different goal activations are bound to be different since different combinations of goals require different optimal solution strategies. Nevertheless it seems intuitively clear that a multi-purpose robot should be capable of encapsulating knowledge in a goal-specific way and transferring experience/knowledge about attaining a goal in one set of goal activations to another.

4.3 The modular architecture

The modular architecture shown in Figure 4 provides this capability. In the modular architecture, instead of maintaining a single monolithic Q-learning system (whose state space size is exponential in the number of goals), a set of independent fixed sized Q-learning modules are used. Each module is itself an adaptive controller and is responsible for learning to achieve a single goal. The state space of each module encodes only information about the external world (goal activation information is not represented). The set of states, actions, and the transition function for each module are taken to be the same, so that the modules differ only in their reward function[6].

Modules learn to achieve their respective goals in parallel. That is, on each step, each module updates its policy and action-value function according to the update

[5] Even if techniques are used that provide for generalization (or interpolation), such as CMACs or neural networks, they are unlikely to be effective since a small change in the activation vector (e.g., the activation/deactivation of a single goal) can profoundly effect the action-value.

[6] Actually, all of our algorithms only assume that there is a common set of actions defined across all modules, allowing for differences in state spaces and transition functions between modules. This might occur, for example, if there are input bits encoding features of the domain that are only relevant to particular goals.

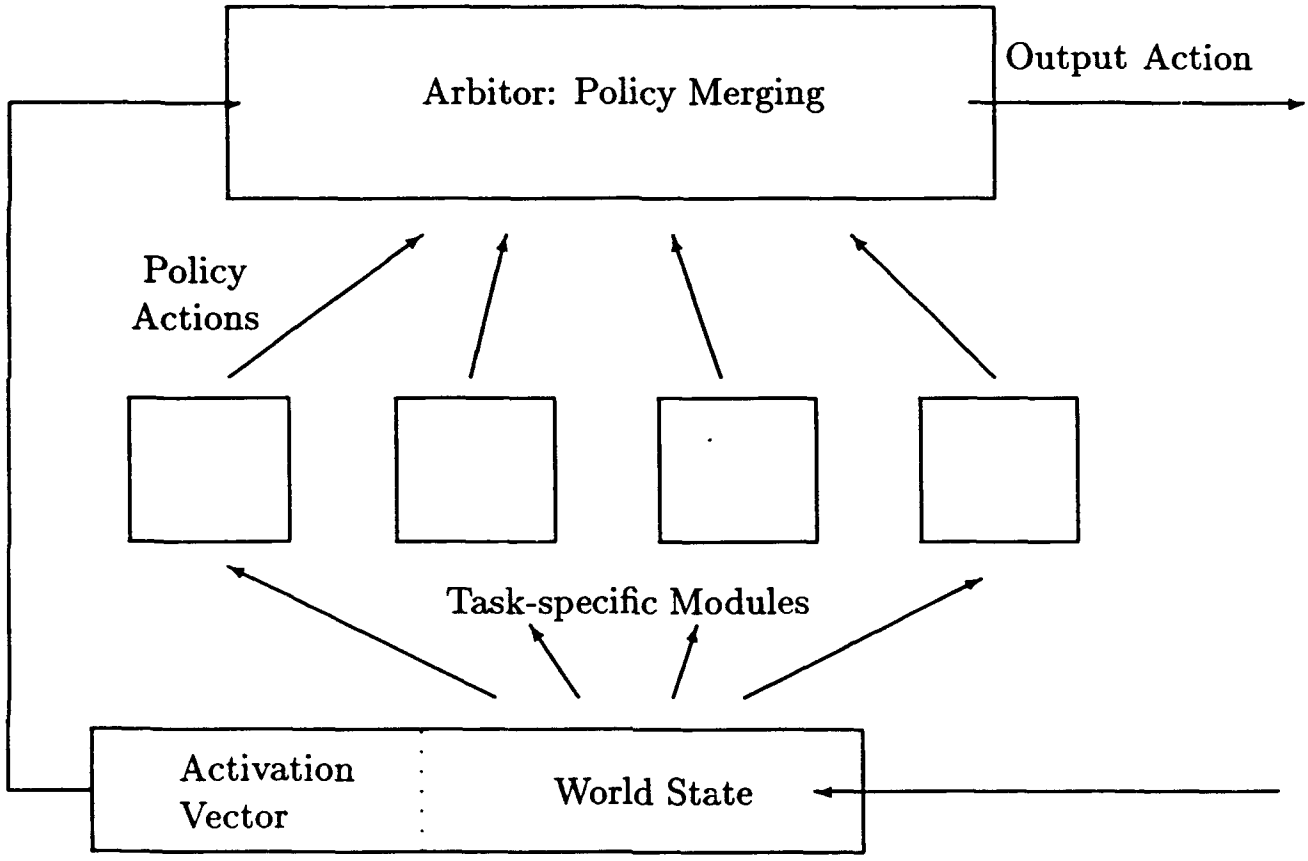

Figure 4: The modular architecture. Each module is responsible for learning to achieve a single goal. The arbiter is responsible for merging information from the individual modules in order to derive the single action performed by the robot at each time step.

rules for Q-learning. Updating is done regardless of whether a module's goal is active. However, each module only receives the reward associated with its goal (cf., Equation 11).[7]

Since individual modules may specify conflicting actions, an arbitration module is used to mediate global control. This arbiter receives as input both the world state and the global activation vector. At each time step, it generates the single action performed by the robot. There are a variety of algorithms that can be used to implement the arbiter. We have focussed on algorithms that construct estimates for the global action-value function, Q_m^*, by merging action-values from the individual modules. A range of merging strategies can be defined by trading off solution quality

[7]We assume that rewards can be distinguished by the goals that generated them and can be routed to the appropriate module.

for computational efficiency. The most elementary strategies combine and compare the modular action-values for the current state only, whereas more sophisticated strategies use a predictive model and lookahead searches to construct more accurate estimates.

4.4 Simple Merging Strategies

We will consider a merging strategy to be simple if

1. only the next action to be executed is computed, as opposed to extended action sequences, and
2. the time to compute the next action is no larger than a small polynomial in the number of goals.

The first condition obviates the need for a detailed predictive model, while the latter condition constrains the kinds of computation that are considered.

One simple merging strategy that we have explored is the *nearest neighbor* strategy, whose global policy we denote by f_{nn}. When using nearest neighbor, the arbiter chooses the action among all active modules that has the maximal action value. That is,

$$f_{nn}(x, \bar{g}) = \arg\max_{a \in A} Q_{nn}(x, \bar{g}, a), \tag{19}$$

where

$$Q_{nn}(x, \bar{g}, a) = \max_{i \in [1 \ldots n]} [Q^i(x, a) \cdot g^i]. \tag{20}$$

and where Q^i denotes the action-value function maintained by the ith module. Returning to the situation illustrated in Figure 3, the robot must choose between going to water and going to food. Assume each module has learned an accurate action value function for its goal, and that the rewards for food and water are identical. Since the food is closest, the policy action associated with the *get-food* module will have the highest action-value, and hence this action will be chosen by f_{nn}. Once food is obtained, the robot will move toward the water, since this is the only remaining active goal. Because each module's policy corresponds to performing a gradient ascent in the utility space, a robot using the nearest neighbor strategy will tend complete its current goal, using the policy from a single module, before switching to another goal and a different module.

Unfortunately nearest neighbor may perform poorly because it considers goals individually. In the case where a cluster of small valued goals has a higher cumulative value than a single large-valued goal, nearest neighbor will prefer the single larger valued goal to the cluster, even when the optimal strategy is to pursue the cluster.

An alternative strategy, which we call *greatest mass*, accounts for goal clustering. Using this strategy, the next action chosen by the arbiter is the one that maximizes the sum of the expected values across all modules. That is,

$$f_{gm}(x, \bar{g}) = \arg\max_{a \in A} \left[Q_{gm}(x, \bar{g}, a) \right], \tag{21}$$

where

$$Q_{gm}(x, \bar{g}, a) = \left[\sum_{i=1}^{n} Q^i(x, a) \cdot g^i \right]. \tag{22}$$

The intuition behind the greatest mass strategy is for the robot to move toward regions of the state space with the most reward. Methods analogous to it have been used in several path planning systems. In [Arbib and House, 1987], a frog's trajectory planning is modeled by letting obstacles and targets (a fly) rate different directions of travel, and chosing the direction with the highest combined rating. Similarly, [Khatib, 1986] assigns positive and negative potentials to obstacles and goals, which combine to form a potential field that the robot traverses using a gradient descent.

With greatest mass, the robot may forgo a large nearby reward in favor of multiple smaller rewards in the future. Generally, the trade off depends upon the number, magnitude, and distance to the rewards and upon the strength of the temporal discount. The less future reward is discounted, the more likely closer goals will be abandoned in favor of a distant cluster. However, for heavily discounted rewards ($\gamma \ll 1$), the greatest mass strategy, like nearest neighbor, will tend to pursue nearby goals first.

These simple strategies have the following desirable characteristics:

1. In each state, arbitration takes time linear in the number of goals.

2. Since only local action values are used for arbitration, simple strategies are appropriate in stochastic domains, where prediction is difficult.

3. Under conditions of high temporal discount ($\gamma \ll 1$), both strategies closely approximate the optimal policy (assuming the individual goal modules have accurate action-value functions).

4.5 Search-based merging

The simple merging strategies described in the previous section only approximate the action-value function for the global task. This follows since the action-values of a given module only encode the expected return given that the robot perform some action, and then follow the policy that is optimal with respect to the module's goal. Whereas, the global action-value function must encode the expected return given that the robot perform some action, and thereafter follow the policy that is optimal with respect to all the goals combined. If we assume that the action-value functions for

the individual modules are accurate, then, in general, Q_{nn} tends to underestimate the global action-value function and Q_{gm} tends to overestimate.

More accurate global action-value estimates, and hence better global performance, can be achieved by using merging strategies that use lookahead search. To facilitate the discussion, assume, for the remainder of this section, 1) that each individual goal module has learned an accurate action-value function for it's goal, 2) that the external environment is completely deterministic, and 3) that the robot has at its disposal a predictive model that it can use to perform mental simulations of possible action sequences. Given these assumptions, one way for the arbiter to proceed is to generate a search tree of possible action sequences; compute (via mental simulation) the utility of each path in the tree; and then select for execution the first action in the sequence with the maximal utility.

More precisely, given a state-action sequence

$$p = \{(x_0, \bar{g}_0), a_0, (x_1, \bar{g}_1), a_1, (x_2, \bar{g}_2), a_2, \\ ...(x_{m-1}, \bar{g}_{m-1}), a_{m-1}, (x_m, \bar{g}_m)\},$$

define its *sequence-value*, denoted $SV(p)$, to be

$$SV(p) = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} R^j(x_i) \cdot g_i^j \right) \gamma^{i-1} \quad + \quad \gamma^m V_m^*(x_m, \bar{g}_m), \tag{23}$$

where V_m^* is the optimal value function for the global task. SV is a generalization of the action-value function Q_m^*, in that it is defined to be the return the robot expects to receive given that starting in state $(x_0, \bar{g}_0)$ it follows the actions specified in path p, and then follows the optimal policy thereafter. For the special case where

$$p = \{(x_0, \bar{g}_0), f_m^*(x_0, \bar{g}_0), (x_1, \bar{g}_1), f_m^*(x_1, \bar{g}_1), (x_2, \bar{g}_2), f_m^*(x_2, \bar{g}_2), \\ ...(x_{m-1}, \bar{g}_{m-1}), f_m^*(x_{m-1}, \bar{g}_{m-1}), (x_m, \bar{g}_m)\},$$

where f_m^* is the optimal global policy, $SV(p) = Q_m^*((x_0, \bar{g}_0), a_0) = V_m^*(x_0, \bar{g}_0)$. Also notice that in general, SV may be difficult to compute even when a predictive model is available. This follows since 1) the activation vector $\bar{g}_i$ is a stochastic function (i.e., new goals may become active while performing the sequence), and 2) V^* is not known *a priori*. Computable approximations to $SV(p)$ can be obtained 1) by assuming new goals do not become active during the action sequence, and 2) by approximating $\gamma^m V^*(x_m, \bar{g}_m)$. Let us define an approximation, $\widehat{SV}$, as

$$\widehat{SV}(p) = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} R^j(x_i) \cdot \hat{g}_i^j \right) \gamma^{i-1} \quad + \quad \gamma^m \hat{V}(x_m, \hat{\bar{g}}_m), \tag{24}$$

where

$$\hat{g}_i^j = \begin{cases} g_0^j & \text{if } i = 0 \\ g_{i-1}^j & \text{if } x_{i-1} \in G^i \\ 0 & \text{otherwise.} \end{cases} \tag{25}$$

and $\hat{\bar{g}} = \hat{g}^1 \cdot \hat{g}^2 \cdot \hat{g}^3 \cdots \hat{g}^n$. Possible candidates for $\hat{V}$ can be obtained from the simple merging strategies defined above. For instance, we may choose

$$\hat{V}(x_m, \hat{\bar{g}}_m) = V_{nn}(x_m, \hat{\bar{g}}_m) = \max_{a \in A} Q_{nn}((x_m, \hat{\bar{g}}_m), a) \tag{26}$$

or perhaps

$$\hat{V}(x_m, \hat{\bar{g}}_m) = V_{gw}(x_m, \hat{\bar{g}}_m) = \max_{a \in A} Q_{gm}((x_m, \hat{\bar{g}}_m), a). \tag{27}$$

Or for m sufficiently large, we may simply choose to ignore the contribution due to $V^*(x_m, \bar{g}_m)$ since it is so heavily discounted (i.e., $\gamma^m V^*(x_m, \bar{g}_m) \approx 0$). If the goal activation rate is small and the length of the sequence p is large, then $\widehat{SV}$ will accurately estimate SV.

Clearly searching all possible sequences for a maximal path is intractable since the search space grows exponentially with path length. Therefore heuristic methods must be used to generate a set of plausible sequences to search. Since the purpose of the arbiter is to select the single action to perform next, a useful set of sequences to search over are those that begin by performing some action $a \in A$ and then follow the optimal global policy until all active goals have been achieved. If we denote this set of sequences by $P^* = \{p_{a_1}, p_{a_2}, p_{a_3}, \ldots p_{a_k}\}$ where $k = |A|$, and p_{a_i} is the sequence that begins with action a_i, then

$$SV(p_{a_i}) = Q^*((x_0, \bar{g}_0), a_i) \tag{28}$$

and $f^*_m(x_0, \bar{g}_0) = \arg\max_{a_i \in A}[SV(p_{a_i})]$. If $\widehat{SV}$, given in Equation 24, is an accurate estimator for SV for $p \in P^*$ then

$$\hat{f}(x_0, \bar{g}_0) = \arg\max_{a_i \in A} \widehat{SV}(P_{a_i}) \tag{29}$$

will yield optimal global performance.

Of course, f^*_m is not known *a priori*, therefore the search set P^* cannot be generated directly. However, if a reasonably good estimate for f^*_m can be obtained, say f', then P' the set of sequences that result from replacing f^*_m with f' when generating P^*, can be used as an alternative.

This idea has lead to an arbitration strategy which we call *Lookahead Nearest Neighbor* (LANN). In LANN the arbiter defines, for each action $a \in A$, p^{nn}_a to be the sequence that results from first performing action a in the initial state $(x_0, \bar{g}_0)$, and then following the nearest neighbor policy, f_{nn}, until each active goal has been achieved. That is,

$$\begin{aligned} p^{nn}_a = \{&(x_0, \bar{g}_0), a, (x_1, \hat{\bar{g}}_1), f_{nn}(x_1, \hat{\bar{g}}_1), (x_2, \hat{\bar{g}}_2), f_{nn}(x_2, \hat{\bar{g}}_2), \\ &\ldots (x_{m-1}, \hat{\bar{g}}_{m-1}), f_{nn}(x_{m-1}, \hat{\bar{g}}_{m-1}), (x_m, \hat{\bar{g}}_m)\}. \end{aligned}$$

If we define $Q_{\text{LANN}}((x, \bar{g}), a) = \widehat{SV}(p^{nn}_a)$ and $f_{\text{LANN}}(x, \bar{g}) = \arg\max_{a \in A} Q_{\text{LANN}}((x, \bar{g}), a)$, then, when $\widehat{SV}$ closely approximates SV, Q_{LANN} will better estimate Q^* than Q_{nn} and

f_{LANN} will be a better global policy than f_{nn}. This follows since Q_{LANN} is constructed by projecting along a path that is nearly optimal with respect to the global task and summing the rewards obtained, whereas Q_{nn} is based on the return estimates associated with pursuing only the nearest (or highest utility) goal and ignoring all other goals and their possible interactions.

4.6 Extending the look ahead strategies

The basic search-based strategies discussed so far can be extended along a number of dimensions. These include:

1. Maintain an estimate of the probability with which the different goals arise. This probability can be used to account for the effects of inactive goals in the sequence values estimates. In might allow the robot to *anticipate* its future need for an inactive goal. For example, using this strategy, if the robot's need for food frequently arose, it would rarely choose actions that brought it far from a known food source.

2. Replace the temporal discount on reward with explicit action costs. In many circumstances, for instance, where each action uses a certain amount of fuel, it is more intuitive to associate fixed costs with each action rather than simply discount future rewards. We have found that when using fixed cost actions and no temporal discounting the optimal strategy is much more likely to involve deferring a more immediate goal in favor of more distant ones.

4.7 A Hybrid Architecture

The monolithic approach described above leads to optimal control in the limit, but has poor initial performance. Conversely, the modular architecture trades off asymptotic performance in favor of faster learning up front. A hybrid architecture that incorporates components of both can achieve both fast learning and optimal asymptotic performance. This architecture has three components: a modular component, a monolithic component, and a modular-monolithic arbiter. The modular component corresponds to a complete modular architecture (see Figure 4), and may use any of the merging strategies described above. The monolithic component is a complete monolithic system (i.e., the monolithic system implements a single, monolithic Q-learning system). The modular-monolithic arbiter (hereafter, the arbiter) is responsible for mediating global control. At each time step, it generates the single action performed by the robot. Early on, we would like the arbiter to select the actions prescribed by the modular component, when it is more likely to yield good performance. However, eventually, we would like the arbiter to rely, more and more, upon the monolithic component.

Unlike the arbiter in the modular component, the modular-monolithic arbiter does not combine utility information from its subcomponents to select an action. Instead, it selects one or the other of the actions proposed by the two sub-components. One selection strategy is simply to choose the action (among the two) with the highest estimated global action-value. That is, associated with the actions proposed by the modular and monolithic components is a global action-value estimate. In the monolithic case, the global estimate is simply the action-value encoded in the monolithic component's action-value function. In the modular component, the global estimate is determined by the particular merging strategy employed. For example, if the nearest-neighbor merging strategy is used, then Q_{nn} can be used. For the greatest mass merging strategy, Q_{gm} can be used. Similar global action-value estimates can be defined for search-based merging strategies.

If the global action-values generated by the modular component are guaranteed to eventually be no greater than the true optimal action-values, then the monolithic system will over time become the dominant component. This follows since the monolithic component will eventually learn the optimal action-value function. However, if the modular component's action-values can overestimate the true global action-values, even when the individual modules have accurate Q-functions, then the monolithic component may never dominate and optimal performance may not be achieved. This follows since an overestimated action-value generated by the modular component may cause the arbiter to select a non-optimal action proposed by the modular component over an optimal action proposed by the monolithic component. If the Q-functions of the individual modules in the modular component accurately estimate the optimal action-values for their respective goals, then Q_{nn} will either equal or underestimate the global action-values. Conversely, Q_{gm} can easily be shown to overestimate in many cases. Thus, for use in a hybrid architecture, the nearest neighbor strategy may be preferred.

4.8 Summary

We have considered control tasks that require the robot to coordinate behavior in order to accomplish a series of independent time varying goals. We have shown how these tasks can be formulated as Markov Decision Processes by augmenting the state space description with a set of goal activation inputs. Formulating the problem in this way leads to an exponential growth in the state space size, and makes the straightforward application of Q-learning intractably slow. To mitigate this scaling problem, we have proposed a modular architecture, which decomposes the task by goals. Each goal is associated with a control module, whose objective is to learn to achieve that goal. Learning is facilitated by reducing the state space from a single monolithic one, whose size is exponential in the number of goals, to a set of fixed sized state spaces (one per goal). Overall control in the modular architecture is the responsibility of an arbiter, who uses information encoded in the individual goal-modules as a resource

for decision making. In general, computationally feasible arbitration strategies yield sub-optimal (but good) performance. Thus, the modular architecture trades off optimal performance in the limit for faster learning and good initial performance. A hybrid architecture that integrates both modular and monolithic components is also described. This architecture exhibits the best performance overall, achieving fast learning initially and optimal performance in the limit.

In the next section, we describe mechanisms whereby the agent can improve its learning time by various social learning mechanisms.

5 Social Learning Mechanisms

When reinforcement learning is used to solve multi-stage decision problems, learning can be viewed as a search process in which the agent, by executing a sequence of actions, searches the world for states that yield reward. For real-world tasks, the state space may be large and rewards may be sparse. Under these circumstances the time required to learn a control policy may be excessive. The detrimental effects of search manifest themselves most a) at the beginning of the task when the agent has an initially unbiased control strategy, and b) in the middle of a task when changes occur in the environment that invalidate an existing control policy.

Experiments have been performed on a series of reinforcement learning systems to study the effect of state-space size on learning rate. The experiments indicate that the learning rate is inversely proportional to the size of the state space for both initial learning and for adapting to change. The experiments further show that the majority of time is spent on the first few trials. During this time, the system performs a quasi-random unbiased (and when adapting to a changed environment, negatively biased) search. Analytical results indicate that while the time required to perform a random search depends strongly on the structure of the state space, for many problems search can be expected to scale exponentially in the size of the state space (independent of the actual difficulty of the task, measured in steps along the optimal solution path).

Two classes of *cooperative learning algorithms* are proposed to reduce search and decouple the learning rate with respect to state-space size. The first, called *Learning with an External Critic* (or LEC), is based on the idea of a mentor, who watches the learner and generates immediate rewards in response to its most recent actions. This reward is then used temporarily to bias the learner's control strategy. The second, called *Learning By Watching* (or LBW), is based on the idea that an agent can gain experience vicariously by relating the observed behavior of others to its own. While LEC algorithms require interaction with knowledgeable agents, LBW algorithms can be effective even when interacting with equally naive peers.

The principle idea being advocated in both of these algorithms is that, in nature, intelligent agents do not exist in isolation, but are embedded in a benevolent society that is used to guide and structure learning. Humans learn by watching others, by being told, and by receiving criticism and encouragement. *Learning is more often a*

transfer than a discovery. Similarly, intelligent robots cannot be expected to learn complex real-world tasks in isolation by trial-and-error alone. Instead, they must be embedded in cooperative environments, and algorithms must be developed to facilitate the transfer of knowledge among agents. Within this context, trial-and-error learning continues to play a crucial role: first for pure discovery purposes, and second, for refining and elaborating knowledge acquired from others.

The ideas for these algorithms were to some extent inspired by Kuniyoshi *et. al.* [Kuniyoshi *et al.*, 1990] who use a robot to recognize plans by watching an operator perform actions. Our own work is more ambitions by incorporating learning algorithms, but at the same time more modest in that many of the important details in a realistic situation have been abstracted away in order to study the essentials of the effects of learning.

Our research involves both empirical and analytical results obtained for these socially inspired learning algorithms. Our empirical study consists of a series of simulation performed on a simple grid world domain and analytical results describe upper bounds on the expected time needed by the LEC and LBW algorithms to first learn the step along the optimal solution path. Results indicate that both algorithms can substantially improve overall performance and in many cases decouple the learning rate from the size and complexity of the state space. Using these mechanisms, the learning rate is more closely related to the difficulty of the task (i.e., the number of steps in the optimal solution), than to the size of the state space. The algorithms are also shown to be robust over a wide range of parameterizations and with respect to interpretation noise.

We also draw similarities between the computational mechanisms required for implementing LEC and LBW algorithms. In particular, the issue of relating observed experiences to ones own is fundamental. We consider initial approaches to this mapping issue.

6 Summary

Our work on real time intelligent problem solving has focussed on the tradeoff between deliberation and activity. Such a tradeoff is required, since an excess of deliberation will be defeated by the dynamical nature of the world and by errors in the predictive model, and a lack of deliberation will not provide the agent with sufficient flexibility to perform well in novel situations. Our framework for evaluating this tradeoff includes both an explicit and an implicit component. In the explicit work, we represent the uncertainties associated with inaccuracies in the model and the inability to completely monitor changes in the world by expanding our language to include probabilities, and making choices about *when* to act and when to deliberate further based upon these explicit uncertainty measures. Ours is the first attempt at developing a language for combining an expressive temporal interval reasoner with probabilities derived explicity from a statistical analysis of the planner's own problem solving activities.

By monitoring the success and failure of the agent's previous choices, the agent is able to improve its predictive model, and hence improve its overall problem solving performance.

In the implicit approach, we use a Markov Decision Process model in order to place a strict time bound on the amount of deliberation. For simple tasks, choosing the next action to perform takes only a constant amount of time via table look-up. The problem with the standard approaches has been an assumption that the agent has immediate, reliable access to all relevant knowledge that might possibly affect its ability to satisfy its goals. One of our main contributions has been in removing this assumption, and modeling an agent without complete knowledge, but rather, where knowledge is obtained through an active sensory system having limited bandwidth. Although the incompleteness of the model results in *perceptual aliasing*, that is, the agent's inability to distinguish between different external world states, the agent is still able to approximate an optimal policy by using its sensors to disambiguate similar states.

With these approaches, we have achieved fast learning on simple tasks, but are faced with intractable learning times for more complex tasks, particularly those that involve multiple goal combinations. In order to improve the learning time, we have investigated two main approaches. The first involves decomposing the controller into a number of task-specific modules, each of which can be learned quickly via our previously described methods. When faced with novel task combinations, however, the agent must use merging strategies, the best of which involve some amount of look ahead using a predictive model. The second approach improves learning by using social mechanisms. Transfer of knowledge among a society of agents can be done either by explicit instruction by a knowledgeable agent to a less knowledgeable agent (learning by external critic), and by agents observing the problem solving activity of one another (learning by watching).

7 Research Activity

The research performed under the IRTPS contract has been widely disseminated in the scientific literature. Overall, this work has been presented at 6 international conferences and workshops, including the Conference on Uncertainty in Artificial Intelligence, the International Machine Learning Workshop, the National Conference on Artificial Intelligence. In addition, researchers have been invited to present aspects of this research at over a dozen universities in the United States, including Dartmouth, MIT, Stanford, and Yale. One of the students has completed his dissertation under this contract (Steve Whitehead – Reinforcement Learning for the Adaptive Control of Perception and Action), and another student is nearing the completion of his dissertation (Nathaniel Martin – Using Statistical Inference to Plan Under Uncertainty). What follows is a list of contract-related publications.

7.1 Publications

Josh Tenenberg, Jonas Karlsson, and Steven Whitehead. "Scaling reinforcement learning through task decomposition," submitted to Second International Conference on Simulation of Adaptive Behavior.

Steven Whitehead, Jonas Karlsson, and Josh Tenenberg. "Learning via task decomposition and policy merging," in *Robot Learning*, Connell and Mahadevan (eds.).

Nathaniel G. Martin and James F. Allen. "A Language for Planning with Statistics", Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence, 1991.

Dana H. Ballard and Steven Whitehead. "Learning Visual Behaviors," in H. Wechsler, editor, *Neural Networks for Human and Machine Perception*, volume II. Academic Press, Boston, 1991.

Steven Whitehead. *Reinforcement Learning for the Adaptive Control of Perception and Action.* PhD Thesis, Department of Computer Science, University of Rochester, Rochester, New York, 1991.

Lambert Wixson. "Scaling Reinforcement Learning Techniques via Modularity," in Proceedings of the Eighth International Workshop on Machine Learning, Evanston, IL, 1991.

Steven Whitehead. "A complexity analysis of cooperative mechanisms in reinforcement learning, In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Anaheim, CA, July 1991.

Steven Whitehead. "A framework for integrating perception, action, and trial-and-error learning." *Special Issue of SIGART on Integrated Intelligent Systems*, July, 1991.

Steven Whitehead. "Complexity and cooperation in reinforcement learning." In *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, IL, June 1991.

Steven Whitehead and Dana Ballard. Learning to perceive and act by trial and error.] *Machine Learning*, 7(1), 1991.

Steven Whitehead and Dana Ballard. "A study of cooperative mechanisms for faster reinforcement learning." TR 365, Department of Computer Science, University of Rochester, 1991.

References

[Allen *et al.*, 1991] Allen, James F.; Kautz, Henry A.; Pelavin, Richard N.; and Tenenberg, Josh D. 1991. *Reasoning About Plans.* Morgan Kaufman Publishing Co., San Mateo, CA.

[Allen, 1983] Allen, James F. 1983. Maintaining knowledge about temporal intervals. *Communication of the ACM* 26(11):832–843.

[Allen, 1984] Allen, James F. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23(2):123–145.

[Allen, 1991] Allen, James F. 1991. Planning as temporal reasoning. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning.* 3–14.

[Arbib and House, 1987] Arbib, Michael A. and House, Donald H. 1987. Depth and detours:an essay on visually guided behavior. In Arbib, M. and Hanson, A., editors 1987, *Vision, Brain, and Cooperative Computation.* MIT Press, Cambridge, MA.

[Barto *et al.*, 1983] Barto, Andrew G.; Sutton, Richard S.; and Anderson, Charles W. 1983. Neuron-like elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics* SMC-13(5):834–846.

[Bellman, 1957] Bellman, R. E. 1957. *Dynamic Programming.* Princeton University Press, Princeton, NJ.

[Bertsekas, 1987] Bertsekas, D. P. 1987. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall.

[Bickel and Doksum, 1977] Bickel, Peter J. and Doksum, Kjell A. 1977. *Mathematical Statistics: Basic Ideas and Selected Topics.* Holden-Day, Inc., Oakland, CA.

[Chapman, 1987] Chapman, David 1987. Planning for conjunctive goals. *Artificial Intelligence* 32:333–377.

[Dean, 1985] Dean, Thomas 1985. *Temporal Imagery: An Approach to Reasoning about Time for Planning and Problem Solving.* Ph.D. Dissertation, Yale University, New Haven, CT.

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

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

[Haddawy, 1990] Haddawy, Peter 1990. Time, chance, and action. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence.* 147–153.

[Haddawy, 1991] Haddawy, Peter 1991. A temporal probability logic for representing actions. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning.* 313–324.

[Hanks, 1988] Hanks, Steven 1988. Representing and computing temporally scoped beliefs. In *American Association of Artificial Intelligence National Conference 1988.* 501–505.

[Holland, 1986] Holland, John H. 1986. Escaping brittleness: the possibilities of general-purpose learning algortihms applied to parallel rule-based systems. In *Machine Learning: An Artificial Intelligence Approach. Volume II.* Morgan Kaufmann, San Mateo, CA.

[Kaelbling, 1990] Kaelbling, Leslie P. 1990. *Learning in Embedded Systems.* Ph.D. Dissertation, Stanford University.

[Kanazawa and Dean, 1989] Kanazawa, Keiji and Dean, Thomas 1989. A model for projection and action. In *Eleventh International Joint Conference on Artificial Intelligence.* 985–990.

[Kanazawa, 1991] Kanazawa, Keiji 1991. A logic and time nets for probabilistic inference. In *American Association of Artificial Intelligence National Conference 1991.* 360–365.

[Khatib, 1986] Khatib, O. 1986. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research* 5(1).

[Koomen, 1989] Koomen, Johannes A.G.M. 1989. Reasoning about recurrence. Computer Science 307, University of Rochester.

[Kuniyoshi *et al.*, 1990] Kuniyoshi, Yasuo; Inoue, Hirochika; and Inaba, Masayuki 1990. Design and implementation of a system that generates assembly programs from visual recognition of human action sequences. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems.* 567–574.

[McCarthy, 1977] McCarthy, John 1977. Epistemological problems of artificial intelligence. In *Proceedings of IJCAI-77.* 1034–1044.

[McDermott, 1982] McDermott, Drew V. 1982. A temporal logic for reasoning about processes and plans. *Cognitive Science* 6:101–155.

[Neyman, 1960] Neyman, J. 1960. *A First Course in Probability and Statistics.* Hold, Rinehaart and Winston, New York.

[Ross, 1983] Ross, S. 1983. *Introduction to Stochastic Dynamic Programming.* Academic Press, New York, NY.

[Sacerdoti, 1974] Sacerdoti, Earl D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5:115–135.

[Schmidhuber, 1990] Schmidhuber, Jurgen 1990. Making the world differentiable: on using self-supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical Report Report FKI-126-90 (revised), Technische Universitat Munchen.

[Shoham, 1988] Shoham, Yoav 1988. *Reasoning About Change.* The MIT Press, Cambridge, MA.

[Sutton, 1990] Sutton, Richard S. 1990. Integrating architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, TX. Morgan Kaufmann.

[Thrun, 1992] Thrun, Sebastian 1992. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, School of Computer Science, Carnegie Mellon University.

[Watkins, 1989] Watkins, Chris 1989. *Learning from delayed rewards.* Ph.D. Dissertation, Cambridge University.

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

[Williams, 1987] Williams, Ronald J. 1987. Reinforcement-learning connectionist systems. Technical Report NU-CCS-87-3, College of Computer Science, Northeastern University, Boston, MA.