Machine Learning, 7, 45–83 (1991)

Learning to Perceive and Act by Trial and Error

STEVEN D. WHITEHEAD (WHITE@CS.ROCHESTER.EDU)
Department of Computer Science, University of Rochester, Rochester, New York 14627

DANA H. BALLARD (DANA@CS.ROCHESTER.EDU)
Department of Computer Science, University of Rochester, Rochester, New York 14627

Editor: Richard Sutton

Abstract. This article considers adaptive control architectures that integrate active sensory-motor systems with decision systems based on reinforcement learning. One unavoidable consequence of active perception is that the agent's internal representation often confounds external world states. We call this phoenomenon *perceptual aliasing* and show that it destabilizes existing reinforcement learning algorithms with respect to the optimal decision policy. We then describe a new decision system that overcomes these difficulties for a restricted class of decision problems. The system incorporates a perceptual subcycle within the overall decision cycle and uses a modified learning algorithm to suppress the effects of perceptual aliasing. The result is a control architecture that learns not only how to solve a task but also where to focus its visual attention in order to collect necessary sensory information.

Keywords. Reinforcement learning, deictic representations, sensory-motor integration, hidden state, non-Markov decision problems

1. Introduction

Recently there has been a resurgence of interest in intelligent control architectures that are based on reinforcement learning methods (RLM) (Barto et al., 1990a; Clocksin & Moore, 1988; Holland, 1986; Miller et al., 1990; Sutton, 1988; Watkins, 1989; Whitehead & Ballard, 1989a; Wilson, 1987). These architectures are appealing because they are both situated and adaptive. Unlike traditional plan-based controllers, RLM systems do not make decisions by appealing to a time consuming search through a space of possible plans. Instead, they maintain a *policy function* that maps situations directly into actions. Decision making reduces to computing the instantaneous value of the policy function and can be performed in constant time—for example, a policy function can be implemented using a table, CMAC, or neural net (all of which can be evaluated in constant time).

The immediacy of decision making puts RLM systems in close relationship with other reactive systems (Agre & Chapman, 1987; Brooks, 1986; Georgeff & Lansky, 1987; Firby, 1987; Drummond, 1989; Nilsson, 1989; Schoppers, 1987). However, RLM systems distinguish themselves from these and most reactive systems in that they are *adaptive*. The vast majority of reactive systems *do not* learn. Instead, their decision knowledge is hand coded into them by their designers, either explicitly (e.g., Agre, 1988; Brooks, 1986; Georgeff & Lansky, 1987; Firby, 1987) or through hand-coded world models which eventually get compiled into a set of reactive rules (e.g., Blythe & Mitchell, 1989; Fikes et

al., 1972; Laird et al., 1986; Schoppers, 1989). RLM systems do not rely on hand-coded decision knowledge. They learn their control strategies by trial and error, by interacting with the world and receiving feedback in the form of rewards. This adaptability relieves the burden of providing complete domain knowledge *a priori* since it is acquired with experience. It also allows the system to adapt to changing circumstances and learn new tasks.

Although RLM systems are promising, to date they have only been applied to relatively simple tasks, such as pole balancing (Barto et al., 1983; Sutton, 1984), simplified navigation (Barto & Sutton, 1981; Booker, 1982; Sutton, 1990a; Watkins, 1989; Wilson, 1987), and easy manipulation games (Anderson, 1989; Whitehead & Ballard, 1989a). Before these systems can be scaled to larger, more complex control problems a number of issues must be addressed. These include developing techniques for improving the learning rate, developing space-efficient implementations of policy and value functions, and incorporating more realistic models of perception and action. Progress on the first two of these issues looks promising (for faster learning see Franklin, 1988; Mahadevan & Connell, 1990; Sutton, 1990b; Whitehead & Ballard, 1989b; for efficient implementations see Girosi & Poggio, 1989; Hormel, 1989). This article deals with the third issue, adopting more realistic models of the agent's sensory-motor system.

The vast majority of work in AI has not dealt realistically with perception, and research in reinforcement learning is no exception. A common simplifying assumption is that a decoupled (often implicit) sensory system automatically provides an embedded decision system with an internal representation that completely describes the state of the external world. This representation frequently takes the form of a set of propositions that describe the relationships between, and the features of, all the objects in the domain. Unfortunately, even for simple toy domains such representations lead to large internal state spaces and unrealistic assumptions about the capabilities of the sensory system. For example, in a classical blocks-world domain containing n blocks, the size of the state space using a traditional representation is $O(n!)$ (Ginsberg, 1989). For $n = 20$ the state space has over forty billion (42,949,672,940) states. Most of the information that distinguishes states in the internal representation is irrelevant to the immediate task faced by the agent and only interferes with decision making (and learning) by clogging the system with irrelevant detail. Furthermore, an overly descriptive representation places undue pressure on the sensory system to maintain its fidelity.

Agre and Chapman have recognized this problem and suggest *deictic representations*, as a more feasible approach based on active sensory-motor systems (Agre & Chapman, 1987; Agre, 1988; Chapman, 1989). The central premise underlying a deictic representation is that the agent need not name and describe every object in the domain, but instead should register information only about objects that are relevant to the task at hand. That is, at any moment the agent's internal representation should register only the features of a few key objects and ignore the rest. Also, those objects should be indexed according to the intrinsic features and properties that make them significant. This approach has three important implications: 1) it leads to compact, task-dependent representations that reflect the complexity of the task instead of the complexity of the domain (which could be arbitrary); 2) it leads to systems that actively control their sensory apparatus since they must tract relevant objects and change their focus of attention as objects come into and fade from significance (Ballard, 1989); and 3) it leads to architectures with feasible sensory-motor

subsystems since perception and action are reduced to a process of finding, tracking, and responding to only a few key objects at a time. In the case of a blocks-world task, the agent might focus on two or three key blocks at a time and be oblivious to the rest (Chapman, 1989).

In this article, we describe our experiences with architectures that incorporate both deictic representations (for feasible perception) and reinforcement learning methods (for adaptive control). In particular, we show that integrating deictic representations (and active perception in general) and reinforcement learning into a single control architecture is non-trivial because the use of deictic representations results in internal states that confuse states in the external world. We term this phenomenon *perceptual aliasing* and show that it can severely interfere with the decision system's ability to learn an adequate control policy. What makes learning in this context difficult is that, in addition to learning the overt actions needed to solve a problem, the agent must also discover how to configure its sensory system (i.e., focus its attention) in order to accurately represent the state of the world with respect to the task. If the agent attends to the few key objects relevant to the task, then its internal state accurately represents the world. If, however, the agent does not attend to those key objects, then the internal state may say nothing useful about the world. A dilemma arises: in order for the agent to learn to solve a task, it must accurately represent the world with respect to the task; but, in order for the agent to learn an accurate representation, it must know how to solve the task.

We approach these issues by focusing on a restricted class of decision problems which we call *problem-solving tasks*. In a problem-solving task, the agent is repeatedly presented with instances of the task (a series of trials). In each trial, the agent is presented with an instance of the problem to be solved (i.e., an initial state). The agent's objective is to execute a sequence of actions that drives the world into a desirable goal state. When the goal is achieved the agent receives a positive reward and the trail ends. If after a predetermined number of steps the agent fails to solve the problem, it gives up and goes on to the next trial.

A new decision system that learns problem-solving tasks has been developed. The decision system embeds a perceptual cycle within the overall decision cycle and uses a modified learning algorithm to eliminate the undesired effects of perceptual aliasing. What makes the decision system unique is that, while learning the overt control strategy needed to solve the task, it simultaneously learns a perceptual control strategy and a task-dependent representation of the world. The system learns incrementally. That is, it first learns to solve and represent very simple instances of the task. The solutions to those instances provide it with enough knowledge to learn to represent and solve slightly more difficult problems. This boot-strapping process repeats itself indefinitely until the agent has learned to represent and solve all instances of the task. The new design does not require any special ordering of problem instances because the learning algorithm is stable (i.e., an inability to solve hard problems does not disrupt the agent's knowledge for solving easy problems) and the agent eventually gives up on problems it deems too hard (Blum & Blum, 1975).

The remainder of the article is organized as follows. Section 2 presents the background material needed for discussing the integration of deictic representations and reinforcement learning. In particular, a formal model of the agent and its environment is presented, the principles of deictic sensory-motor systems are elaborated, and the essentials of reinforcement learning are reviewed. Perceptual aliasing and its impact on learning are discussed

in Section 3. In particular, we introduce the notion of *inconsistent internal states* and show how they destabilize reinforcement learning algorithms with respect to the optimal policy. In Section 4, the new decision system is described. An example of the new system is given in Section 5, where we describe a program that learns to solve a simple block manipulation task. A discussion of the system's limitations and future prospects can be found in Section 6, and the conclusions are drawn in Section 7.

2. Foundations

2.1. Embedded learning systems

Before going into the details of deictic representations, reinforcement learning, and perceptual aliasing, it is useful to formalize concepts such as "the world," "the agent," "the sensory-motor system," and "the decision system." For this purpose we begin by adopting a formal model for describing embedded learning systems. The model, shown in Figure 1, extends a model proposed by Kaelbling (1989) by explicitly representing the dynamic relationship between external world states and the agent's internal representation.

2.1.1. The external world

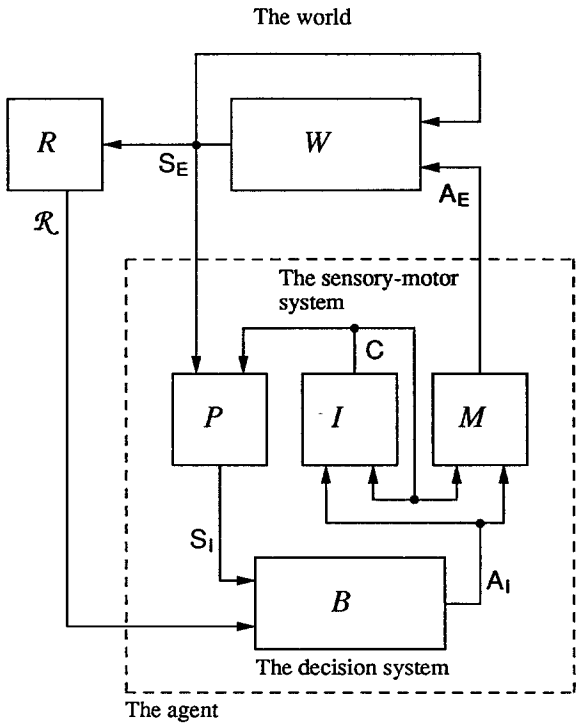
The world is modeled as a discrete time, discrete state, Markov decision process (Bellman, 1957; Ross, 1983; Bertsekas, 1987) and is described by the tuple $(S_E, A_E, \mathcal{W}, \mathcal{R})$, where S_E is the set of world states, and A_E is the set of physical actions that can be executed by the agent. Time is modeled by the discrete sequence, $t = 0, 1, 2, \ldots$, and a clock tick occurs whenever a physical action is executed. $\mathcal{W}$ is a state transition function that maps the current state and an action into the next world state (i.e., $\mathcal{W} : S_E \times A_E \rightarrow S_E$). In general, transitions can be probabilistic so that $\mathcal{W}(s, a)$ may return a sample state from a probability distribution over S_E. The probabilities that govern the transition function depend only upon the current state and the action and are denoted by $P_{x,y}(a)$, where

$$P_{x,y}(a) = Pr(\mathcal{W}(x, a) = y). \tag{1}$$

$\mathcal{R}$ is the reward function and maps world states into real valued rewards (i.e., $\mathcal{R} : S_E \rightarrow \Re$). As with $\mathcal{W}$, $\mathcal{R}$ is probabilistic and can be described by a reward distribution function $P_{rwd}(x, r)$, where

$$P_{rwd}(x, r) = Pr(\mathcal{R}(x) = r). \tag{2}$$

Rewards associate value to individual states and are used by the agent to improve performance. Positive rewards indicate that the world is in a desirable state and negative rewards indicate undesirable states.[1,2]



Symbol	Function Name	Mapping	Probabilistic	Adaptive
W	Transition	$S_E \times A_E \rightarrow S_E$	Yes	No
R	Reward	$S_E \rightarrow \mathcal{R}$	Yes	No
P	Perceptual	$S_E \times C \rightarrow S_I$	No	No
I	Configuration	$A_I \times C \rightarrow C$	No	No
M	Motor	$A_I \times C \rightarrow A_E$	No	No
B	Behavioral	$(S_I \times \mathcal{R})^* \rightarrow A_I$	Yes	Yes

Figure 1. A formal model for an agent with an embedded learning subsystem and an active sensory-motor subsystem. The table summarizes the functions implemented by each of the model's modules.

2.1.2. *The agent's task*

Roughly, the aim of the agent is to maximize the reward it receives over time. The agent does not want to merely maximize its immediate reward in the current state: it wants to maximize the cumulative reward it receives over some period of time in the future. Several

measures of cumulative reward can be used. We will assume that the agent tries to maximize a total discounted sum of the reward it receives over time. This sum will be called the *return* and for time t is defined as

$$\mathbf{r}_t = \sum_{n=1}^{\infty} \gamma^{n-1} r_{t+n} \tag{3}$$

where r_t is the reward received at time t, and γ is a discount factor between 0 and 1.

The agent's overt behavior can be characterized by a policy function π_E, which maps states into actions ($\pi_E : S_E \rightarrow A_E$). If the world is in state s and the agent follows the policy π_E indefinitely, then the agent's expected return is denoted by $V_{\pi_E}(s)$. V_{π_E} is called the *value function* for policy π_E, and $V_{\pi_E}(s)$ is called the *utility* of state s. Formally, $V_{\pi_E}(s)$ is defined as

$$V_{\pi_E}(s) = \mathbf{E}[R(s, \pi_E, 1) + \ldots \gamma^{n-1} R(s, \pi_E, n) + \ldots] \tag{4}$$

where $R(s, \pi_E, n)$ is the random variable denoting the reward received at time $t + n$, given that at time t the system is in state s and follows π_E for n-steps.[3] If a *decision* is defined as the act of choosing to execute an action in a given state, and is denoted by the state-action pair (s, a), then a function that associates a value with each possible decision can be defined. This function, called the *action-value function*, is denoted as Q_{π_E} for policy π_E and is defined as the expectation of the return the system will receive given that it executes action a in state s and follows policy π_E thereafter. That is,

$$Q_{\pi_E}(s, a) = \mathbf{E}[\mathfrak{R}(S(s, a)) + \gamma V_{\pi_E}(\mathfrak{W}(s, a))]. \tag{5}$$

For stationary Markov decision processes it can be shown (Bellman, 1957; Ross, 1983) that an optimal policy, denoted π_E^*:

1. is deterministic
2. uniformly maximizes the value function over all states—that is, for all $s \in S_E$,

$$V_{\pi_E^*}(s) = \max_{\pi}(V_{\pi}(s)) \tag{6}$$

3. corresponds to the set of decisions that for each state maximizes the action-value function—that is, for all $s \in S_E$,

$$\pi_E^*(s) = a \quad \text{such that} \quad Q_{\pi_E^*}(s, a) = \max_{b \in A_E}(Q_{\pi_E^*}(s, b)). \tag{7}$$

The agent's objective is to learn and implement an optimal decision policy.

2.1.3. *Problem-solving tasks*

In this article, attention is focused on a restricted class of Markov decision problems, which we term *problem-solving tasks*. In a problem-solving task, the following restrictions apply:

- The world is completely deterministic (i.e., $\mathcal{W}$ and $\mathcal{R}$ are deterministic functions).
- The reward function is uniformly zero for all states except a set of distinguished goal states in which the agent receives a fixed positive reward.
- The task is organized into a series of trials, where each trial begins when the agent is presented with a new instance of the problem and ends when either a goal state is achieved or a time limit expires.
- After each trial, the agent's return is reset so that the return associated with each trial depends only upon the time it takes the agent to solve the immediate problem instance.

Although they represent a restricted class of Markov decision problems, problem-solving tasks (or problems similar to them) are fairly common in the reinforcement learning/adaptive control literature (Anderson, 1989; Barto et al., 1983; Sutton, 1990a; Whitehead, 1989; Yee et al., 1990). Problem-solving tasks are the focus here for two reasons: 1) they are simple, yet sufficient to demonstrate the difficulties caused by perceptual aliasing; and 2) the new learning algorithm described below exploits the reliability of deterministic worlds to deal with perceptual aliasing.[4] The fact that the agent can quit and go on to a new instance of the task if it fails to solve a problem after a sufficiently long period of time is an important feature of our task definition. It allows the agent to indirectly filter difficult instances (and avoid the long searches associated with them) and focus on solving simpler instances first. The agent learns to solve difficult instances through an incremental bootstrapping process that is more efficient than when quitting is not allowed.

2.1.4. *The agent*

Our model of the agent has two major subsystems: a sensory-motor subsystem and a decision subsystem. The sensory-motor subsystem implements three functions: 1) a perceptual function $\mathcal{P}$; 2) an internal configuration function $\mathcal{G}$; and 3) a motor function $\mathcal{M}$. The purpose of the sensory-motor subsystem is to ground internal perceptions and actions in the real world. On the sensory side, the system translates the world state into the agent's internal representation. Since perception is active, this mapping is dynamic and dependent upon the configuration of the sensory-motor apparatus. Formally, let S_I be the finite set of possible internal states, and C be the (possibly infinite) set of sensory-motor configurations. Then, the relationship between external world states and the agent's internal representation is modeled by the *perceptual function* $\mathcal{P}$, which maps world states S_E and sensory-motor configurations C onto internal representations S_I (i.e., $\mathcal{P} : S_E \times C \rightarrow S_I$). On the motor side, the agent has a finite set of *internal motor commands*, A_I, that affect the model in two ways: they can either change the state of the external world (by being translated into external actions, A_E), or they can change the configuration of the sensory-motor subsystem. Internal commands that change the state of the external world are called overt actions

and are denoted by the set A_O, whereas commands that change just the configuration of the sensory-motor system are called perceptual actions and denoted by the set A_P. As with perception, the configuration of the sensory-motor system relativizes the effects of internal commands. This dependence is modeled by the functions $\mathfrak{M}$ and $\mathfrak{I}$, which map internal commands and sensory-motor configurations into actions in the external world and into new sensory-motor configurations, respectively (that is, $\mathfrak{M} : A_I \times C \rightarrow A_E$ and $\mathfrak{I} : A_I \times C \rightarrow C$).

The other component in the agent's architecture is the decision subsystem. This subsystem is like a homunculus that sits inside the agent's head and controls its actions. On the sensory side, the decision subsystem has access only to the agent's internal representation, not to the state of the external world. Similarly, on the motor side, the decision subsystem generates internal action commands that are interpreted by the sensory-motor system. Formally, the decision subsystem implements a behavior function $\mathfrak{B}$ that maps sequences of internal states and rewards $(S_I \times \Re)^*$ into internal actions, A_I.

In the vast majority of reinforcement learning systems, the sensory-motor subsystem and the dynamic relationship it maintains between the world and the agent's internal representation is not modeled explicitly. Instead, the decision system is coupled directly to the world and has complete knowledge of the world state. In contrast, the decision problem facing our decision subsystem is *not* the same as the general problem facing the agent. In general, the decision subsystem's objective is to learn a control policy that takes as inputs the agent's internal representation and generates internal action commands, which when translated correspond to optimal actions in the world. The decision subsystem has the additional task of controlling the agent's sensory-motor system, which it must exploit to gain knowledge about the external world.

2.2. Reinforcement learning

The task faced by the *agent* is representative of learning problems that have previously been studied in reinforcement learning: that is, given the current state, a set of possible actions, and previous trial and error experience, choose the best next action. As will be seen in the next section, classical reinforcement learning algorithms cannot be directly applied to problems in which the decision system's access to the world is modulated by a limited (albeit dynamic) sensory-motor system. Nevertheless, a brief review of reinforcement learning is in order since our eventual design is based directly on those classical approaches. For this subsection we will temporarily ignore the sensory-motor interface and neglect the distinction between the world (S_E, A_E) and the decision system's view of it (S_I, A_I). In our experiments we have focused on a representative learning algorithm known as *Q-learning*, and our brief review follows the development in Watkins (1989). However, our analysis applies virtually all reinforcement learning algorithms that use temporal difference methods to solve the temporal credit assignment problem (Sutton, 1988). A more thorough treatment of Q-learning can be found in Watkins (1989), and reviews of reinforcement learning in general can be found in Barto et al., (1990b); and Williams, (1987).

In Q-learning the agent maintains an action-value function of its own. For time t this function is denoted Q_t. The agent's action-value function is intended to estimate the action-value function of the optimal policy (i.e., $Q \approx Q_{\pi_E^*}$ and, hopefully, $\lim_{t\rightarrow\infty} Q_t = Q_{\pi_E^*}$).

Given Q, the agent's policy, denoted π_t for time t, is determined by analogy with Equation 7. For all $s \in S_E$,

$$\pi_t(s) = a \quad \text{such that} \quad Q_t(s, a) = \max_{b \in A_E}(Q_t(s, b)). \tag{8}$$

That is, for a given state s, the policy function simply selects the action that, according to Q_t, maximizes the expected return.

In Q-learning, the action-value function is estimated by keeping track of actual returns received after making a decision. Recall from Equation 3 that the *return* at time t is defined as

$$\mathbf{r}_t = \sum_{n=1}^{\infty} \gamma^{n-1} r_{t+n} \tag{9}$$

Because $\gamma < 1$, γ^n will approach zero as n becomes large, and because rewards are assumed to be bounded, for each value of γ there will be some number of time-steps n after which the remaining part of the actual return will be negligible. Hence, the agent may calculate an acceptable estimate of the actual return after n time-steps. To obtain an even better estimate, the agent may correct for the terms that are discarded by adding in (appropriately discounted) the return the system expects to receive starting from time $t + n$. Watkins (1989) refers to this type of estimate as the *corrected n-step truncated return* and defines it as

$$\mathbf{r}_t^{(n)} = \sum_{k=1}^{n} \gamma^{k-1} r_{t+k} + \gamma^n V_t(x_{t+n}) \tag{10}$$

where $V_t(x_{t+n})$ is the agent's estimate of the return it expects to receive starting from state x_{t+n}. V_t estimates the optimal value function based upon the agent's current action-value function and is given by

$$V_t(x) = Q_t(x, \pi_t(x)) \tag{11}$$

Given $\mathbf{r}_t^{(n)}$, the action-value function can be updated using the following *n-step Q-learning* rule:

$$Q_{t+n}(x_t, a_t) = Q_t(x_t, a_t) + \alpha(\mathbf{r}_t^{(n)} - Q_t(x_t, a_t)). \tag{12}$$

Here $\mathbf{r}_t^{(n)} - Q_t(x_t, a_t)$ is an estimate of the error in Q_t that is based on watching the reward received over the next $n - 1$ steps, and α is a constant that affects the learning rate. Actually, Equation 12 defines a family of learning rules, one for each n. For large n, the agent waits for the future to unfold before updating Q_t and does not heavily rely on $V_t(x_{t+n})$. Conversely, for small n, the agent updates Q_t after a short delay and relies heavily on $V_t(x_{t+n})$ to accurately predict future reward.[5]

For the remainder of the article, when it is convenient and when there is no possibility of confusion, the explicit time subscripts will be dropped from Q, π, and V. Under these circumstances, it is important to remember that these functions are estimates maintained by the agent that change over time and with experience.

Figure 2 outlines a simple but representative decision/learning cycle for a decision system based on 1-step-Q-learning. The first step in the cycle is to select the next action for execution. With probability p, the system selects the action specified by its control policy $\pi(x)$; otherwise, it chooses an action at random. The action is then executed and the subsequent state and reward are noted. Once the effects of the action are known, the error in the action-value function for the current decision is computed and used to update Q. Finally, $\pi(x)$ and $V(x)$ are updated to reflect changes in Q. The reason the decision system does not always select the action specified by its policy is that the action-value of a decision is only updated when that decision is executed. Occasionally, selecting a random action ensures that each decision will be evaluated periodically. Because the action-value of a decision is updated after a 1-step delay, 1-step-Q-learning is particularly simple. Nevertheless, the learning rule is effective. Watkins has shown that under standard assumptions for Markov decision processes, decision systems based on 1-step-Q-learning, using an appropriate exploration strategy and an appropriately decreasing learning rate, are guaranteed to learn an optimal decision policy (Watkins, 1989). Even though the algorithm in Figure 2 will learn the optimal policy ($\pi \rightarrow \pi_{E*}$) for any problem solving task, the control algorithm will not perform optimally since with probability $1-p$ the system chooses a random action. This inconvenience can be improved upon by adopting a slightly more complex procedure for controlling exploration (Barto et al., 1990).

Example decision cycle for 1-step Q-learning:

1) Generate a random number q between 0.0 and 1.0
2) If $(q < p)$
   then $action \leftarrow \pi(x)$, where π is the policy function and x is the current state
   else $action \leftarrow R(A_E)$, where $R()$ is a random selection function.
3) Execute $action$, let x_{new} be the resulting state and r be the reward received.
4) Compute the 1-step error:
   $error \leftarrow [r + \gamma V(x_{new})] - Q(x, action)$
5) Update the action-value of the selected decision:
   $Q(x, action) \leftarrow Q(x, action) + \alpha error$
6) Update the decision policy (for state x):
   $\pi(x) = a$ such that $Q(x, a) = \max_{b \in A_E}[Q(x, b)]$
7) Update the evaluation function (for state x):
   $V(x) \leftarrow Q(x, \pi(x))$
8) Update the current state: $x \leftarrow x_{new}$
9) Go to 1.

Figure 2. The steps in the decision cycle of a system based on 1-step Q-learning.

2.3. Deictic representations

Although the above formal model of the sensory-motor system admits a variety of designs, our research has been motivated directly by the work of Agre and Chapman on *deictic representations*[6] (Agre & Chapman, 1987; Agre, 1988; Chapman, 1989). This subsection discusses the essential ideas behind deictic representations and describes the deictic sensory-motor system we used in a program that learns a simple block manipulation task.

A central concept underlying deictic representations is the *marker*, around which nearly all perception and action revolve.[7] Markers are best thought of as pointers implemented by the sensory-motor system: ideally, a marker points at an object in the world and registers features of that object in the internal representation.[8] We will describe a marker as *bound* to an object if the marker is pointing to it. A marker can be bound to only one object at a time, and it is assumed that the sensory-motor system maintains the marker's binding at all times. Changing a marker's binding is accomplished by executing explicit actions specifically targeted for that marker. These actions index target objects in the world according to specific features that distinguish them from other objects. For example, a system might have a marker M_1 and an associated action *Move-M_1-to-Red*, which is used to index and mark red objects. In this case, executing *Move-M_1-to-Red* causes the sensory-motor system to search the world for a red object and bind M_1 to it. If a red object cannot be found, the action fails and M_1's binding remains unchanged. If multiple red objects exist, the sensory-motor system chooses the first one it comes to.

With respect to our formal model (Section 2.1.4), the configuration of the agent's sensory-motor system is defined by its marker bindings since knowledge of those bindings, along with knowledge of the world state, is sufficient to determine the values of the bits in the agent's internal representation.

In a deictic representation the agent's sensory inputs fall into three general categories: peripheral aspects, local aspects, and relational aspects. *Peripheral aspects* register general, spatially non-specific information about the world, such as the presence or absence of certain colors, shapes, and motions. Both local aspects and relational aspects register properties of marked objects. *Local aspects* register intrinsic, local features of a marked object, such as its shape, color, and texture. *Relational aspects* register relational properties between marked objects, such as relative shape, relative color, and relative position. The moment-by-moment values of these three sets of inputs define the agent's internal representation.

A key feature of deictic representations is that there are only a limited number of markers, say less than ten (the system described below has two markers). The small number of markers and the limited number of features associated with each marker keep both the internal representation and the number of possible actions much smaller than is possible with conventional representations. If an object in the world is not marked, then it is invisible to the system (except for the effects it registers in the periphery). The emphasis is on keeping the internal representation small and task-specific. Also, because the sensory-motor system is active, the system can dynamically track the objects that are relevant and change its focus of attention (marker bindings) as these objects come into and fade from significance.

Markers also play an important role in motor control since overt actions are predominately specified with respect to them. In this case, a marker's binding acts to establish the reference

frame in which an action is performed. For example, the overt action *Place-at-M_1* might cause the agent to place an object it is holding at the location currently pointed to by marker M_1. We distinguish two types of markers: *overt* markers and *perceptual* markers. A marker is overt if it has an action associated with it that affects the state of the external world. Otherwise, it is a perceptual marker. Overt markers are used for establishing reference frames for actions in the world, while perceptual markers are used for collecting additional information about the current state. Actions associated with overt markers are called *overt actions* and actions associated with perceptual markers are called *perceptual actions*.[9]

As an example, Figure 3 lists the specifications for the deictic sensory-motor system used by a program (to be described later) that learns to solve a simple block manipulation task. The system has two markers: an *action marker* and an *attention marker*. The action marker is used for both perception and action, while the attention marker is used only for perception. Each marker has a set of local aspects associated with it; these report the color and shape of the marked object, the number of blocks stacked above the marked object, whether or not the marked object is sitting on the table, and whether or not the marked object is being held by the robot. The system has two relational aspects—one for recording vertical alignment between the two markers and one for recording horizontal alignment. Peripheral aspects include inputs for detecting the presence of colors in the scene (red, green, and blue) and for detecting whether the robot is currently holding an object.

The internal motor commands available to the decision subsystem are shown on the right in Figure 3. In this example, all overt actions are made with respect to the action marker. The two primary overt actions are for grasping and placing objects. For grasping, the action *grasp-object-at-action-marker* causes the robot to pick up the object marked by the action marker. The action works if the robot's hand is empty and the marked object has a clear top. Similarly for placing, the action *place-object-at-action-frame* causes the system to place a block it is holding on top of the object pointed to by the action marker. This action works if the robot is holding a block and the target object has a clear top. Other overt actions include commands for moving the action marker. Although these may appear to be perceptual actions, they are overt actions in the strictest sense because they affect the robot's ability to perform other overt actions.

The attention marker is a perceptual marker and has a repertoire of perceptual actions that are used exclusively for gathering additional sensory information. As will be seen in Section 4, the attention marker plays an important role in allowing the system to disambiguate world states.

All told the sensory-motor system has a 20-bit input vector (See Figure 3, left): 4 bits of peripheral aspects, 14 bits of local aspects, 2 bits of relational aspects; and 14 actions: 8 overt and 6 perceptual.

Notice that the internal state space defined by the sensory inputs is small compared to the state space that could result if every object in the domain were represented. The principal advantage is that this reduction leads to more feasible perception and simpler decision tasks. The principal disadvantage is that it limits the complexity of the problems that can be solved by the agent. For example, if during the course of a problem, a decision depends upon features of three separate blocks, then an agent with the above sensory-motor system will not be capable of solving the problem because it cannot simultaneously represent features of more than two blocks. Of course, the sensory-motor system could be

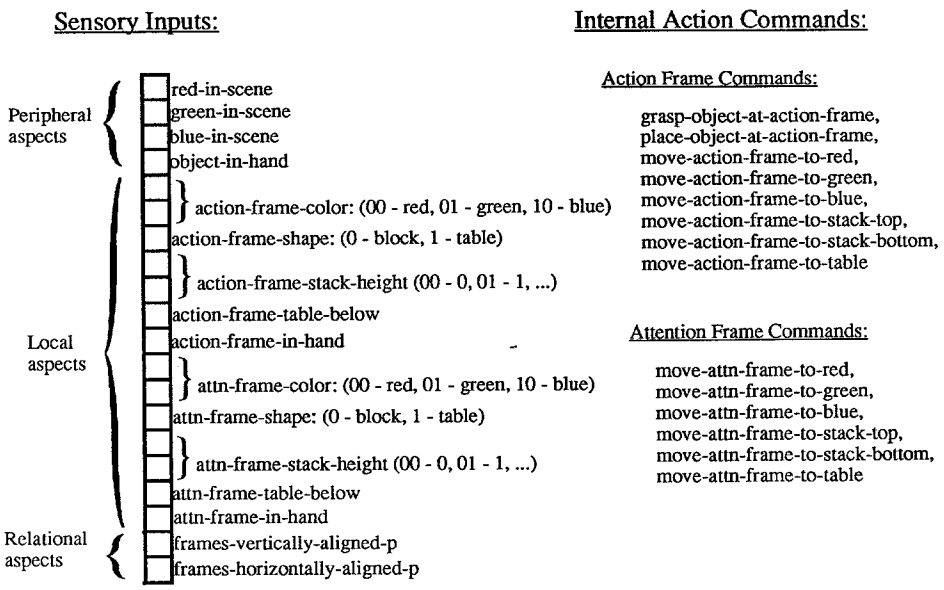


Figure 3. The specification for a deictic sensory-motor system containing two markers. The system has a 20-bit input vector, 8 overt actions, and 6 perceptual actions. The values registered in the input vector and the effects of internal action commands depend upon the binding between markers in the sensory-motor system and objects in the external world.

expanded to allow the system to register features of three blocks (for example, by adding an additional marker), but in general new problems can always be defined that are beyond the scope of the current system. Our contention is that many of the problems we are interested in solving (or learning to solve) only involve keeping track of a few objects at a time (for example, see (Chapman, 1989)).

Also, notice that individual objects in the world are referenced not by arbitrarily assigned names, but by the features that make them relevant. For example, the action *Move-action-marker-to-stack-top* would cause the action marker to move upwards from its current position until it reaches the block at the top of the stack. What makes this top block significant is not any absolute name like "BLOCK-43," but the relationship it holds with the rest of the world. Namely, this block is at the top of a stack and affords (Gibson, 1979) being removed and placed on the table (possibly to get at another more important block). The variety of features and properties that can be used as indices also delimits the types of problems that an agent can solve.

Finally, notice that physical action in the world (e.g., picking and placing blocks) occurs relative to the reference frame defined by the action marker. This is consistent with the view that objects in the world fill roles according to their features and that the control strategy learned by the decision system is specified in terms of those abstract roles.

3. Perceptual aliasing

The straightforward integration of deictic representations and reinforcement learning leads to undesirable interactions that prevent the decision subsystem from learning an optimal control strategy. These interactions arise because the mapping between world states and the agent's internal representation is many-to-many. That is, a state $s \in S_E$ in the world, depending upon the configuration of the sensory-motor subsystem, may map to several internal states; conversely, a single internal state, $s' \in S_I$, may represent multiple world states. We call this overlapping between the world and the agent's internal representation *perceptual aliasing*. Figure 4 illustrates perceptual aliasing in a simple blocks-world domain, where we have adopted the deictic sensory-motor system defined in Figure 3. Figure 4a shows two different world states (top) that generate the same internal representation simply because the markers are focused on parts of the world that are similar. In the figure, the (+) represents the action-frame marker and the (*) represents the attention-frame marker. Similarly, Figure 4b shows that a single world state (block configuration) can produce multiple internal representations, depending upon the placement of the markers.

Perceptual aliasing has a devastating impact on the decision subsystem's ability to learn an adequate control policy because it causes the system to confound world states that it must necessarily distinguish in order to solve the task. The easiest way to illustrate the problem is to consider the effect of perceptual aliasing in a simple problem-solving task. Consider the task whose transition diagram is shown in Figure 5a. In this task, the world has eight states, $S_E = \{s_0, s_1, s_2, \ldots s_6, g\}$, and there are two overt actions, $A_E = \{a_l, a_r\}$. The goal of the task is to enter state g, whereupon the agent receives a fixed reward, $\mathcal{R}(g) = 5000$. Non-goal states yield zero reward, $\mathcal{R}(s_k) = 0$ for $k = 0$ to 6.

At this point, there are two decision problems that must be distinguished: the decision problem faced by the *agent* and the decision problem faced by the embedded *decision subsystem*. The problem faced by the agent is the original problem-solving task defined by the world. The problem faced by the decision subsystem corresponds to the original problem as transformed by the sensory-motor interface. We call these the *actual* (or external) problem and the *perceived* (or internal) problem, respectively.

For problem-solving tasks, the optimal value function, denoted V_E^*,, is an exponentially decreasing function of the distance to the goal. That is, $V_E^*(s) = \mathcal{R}(g)\gamma^{(d(s)-1)}$, where $\gamma < 1$, $\mathcal{R}(g)$ is the reward the agent receives upon entering the goal state, and $d(s)$ is the distance (number of steps) from state s to the goal. The optimal policy, π_E^*, corresponds to choosing the action that minimizes the distance to the goal. When faced with an instance of the problem, the optimal solution trajectory corresponds to performing a gradient ascent of V_E^*. For the actual problem-solving task given above, the optimal policy corresponds to moving right (a_r) at every opportunity (i.e., for all $s \in S_E$, $\pi_E^*(s) = a_r$), and the optimal solution to a given trial corresponds to a trajectory where $V_E^*(x_t)$ is monotonically increasing in time. This result is illustrated in Figure 6a, which plots $V_E^*(x_t)$ versus time for a trial that begins in state s_0 at time $t = 0$ and follows the optimal trajectory to g at time $t = 7$.

If the agent's sensory-motor subsystem is transparent and gives the decision subsystem direct access to the actual decision problem (i.e., $S_I = S_E$ and $A_I = A_E$), then the decision subsystem could learn the *actual task* directly. In general, however, the decision problem

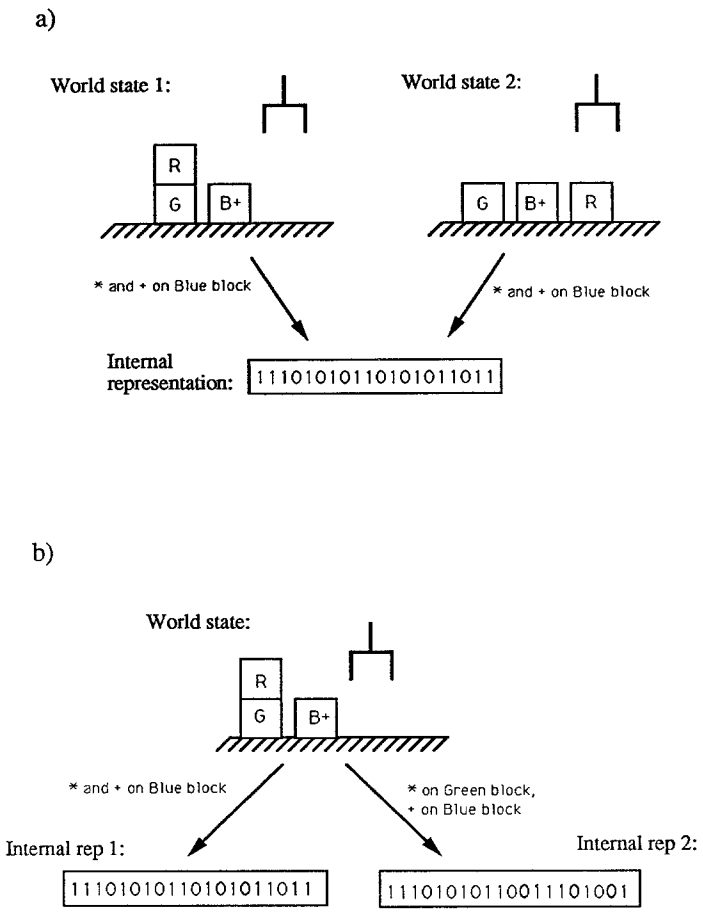


Figure 4. Generally the mapping between external world states and the agent's internal representation is many to many. a) shows how two different external states can generate the same internal representations and b) shows how one external state may have more than one internal representation. In the figure, the (+) represents the action-frame marker and the (*) represents the attention-frame marker.

seen by the embedded decision subsystem, the perceived problem, depends upon the mapping defined by the sensory-motor subsystem. For the task shown in Figure 5a, consider an agent with a sensory-motor subsystem that implements a mapping between the world and the agent's representation that is fixed, one-to-one, and onto except for states s_2 and s_5, which get mapped to the same internal state $s'_{2,5}$. That is, let $S_I = \{s'_0, s'_1, s'_{2,5}, s'_3, s'_4, s'_6, g'\}$, where except for $s'_{2,5}$, s'_j (and g') represents world state s_j (and g). Also let $A_I = \{a'_l, a'_r\}$, where a'_l and a'_r map to a_l and a_r, respectively. The transition diagram describing the perceived decision problem is shown in Figure 5b. Note that this decision problem is not Markovian since the effects of actions are not independent of the past but depend upon the hidden, unperceived state of the actual, underlying decision problem. For example,

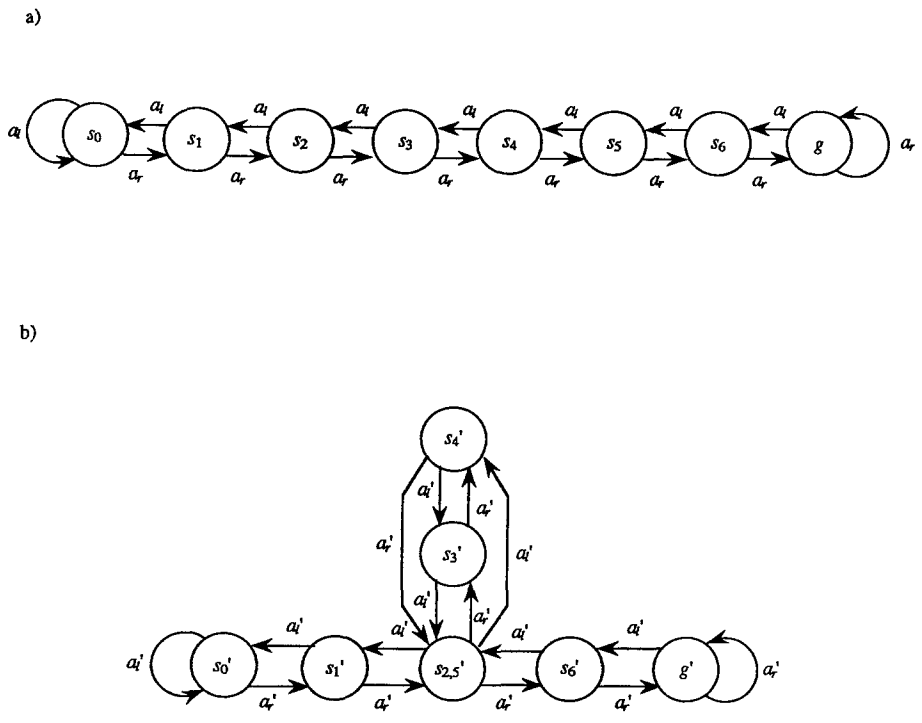


Figure 5. Transition diagrams for a simple problem-solving task: a) the diagram for the actual decision problem, b) the diagram for the perceived decision problem when interpreted through a sensory-motor system with perceptual aliasing.

of the actual, underlying decision problem. For example, the optimal trajectory from s_0' to g' is the sequence s_0', s_1', $s_{2,5}'$, s_3', s_4', $s_{2,5}'$, s_6', g' and contains $s_{2,5}'$ twice—once when the world is in state s_2 and once when it is in state s_5. However, the embedded decision system will never see the sequence s_0', s_1', $s_{2,5}'$, s_6', g'.

This system does not have a stable decision policy. If the policy for the decision system is initialized to the optimal policy and the control rule is fixed so that the system follows the optimal policy with probability $p = 0.99$ and chooses a random action otherwise, and if the decision system is run for many trials and allowed to estimate the value and action-value functions (denoted V_I and Q_I, respectively) then we observe the following. First, since the value and action-value functions are based on *expected* returns, for the state $s_{2,5}'$, they take on values somewhere between the corresponding values for s_2 and s_5 in the actual decision problem. That is,

$$V_E^*(s_2) \leq V_I(s_{2,5}') \leq V_E^*(s_5), \tag{13}$$

$$Q_E^*(s_2, a_r) \leq Q_I(s_{2,5}', a_r') \leq Q_E^*(s_5, a_r), \tag{14}$$

and

$$Q_E^*(s_2, a_l) \leq Q_I(s_{2,5}', a_l') \leq Q_E^*(s_5, a_l). \tag{15}$$

overestimates the expected return at $t = 2$. Similarly, the second time $s'_{2,5}$ is encountered, when $t = 5$ and the world is in state s_5, $V_I(s'_{2,5})$ underestimates the expected return.

If we relax our hold on the decision policy and allow the system to adapt, we find that the optimal policy is unstable! Not only is the system unable to find the optimal policy, it actually moves away from it. In general, it can be shown that the system will oscillate among policies, never finding a stable one. The instability can be understood by considering the effect of the aberrational maximum on the policy. According to Equation 8 the system locally adjusts its policy in order to maximize the expected return. Thus, in state s'_3 the policy will be changed so that the system tends to take actions that move it back to $s'_{2,5}$ instead of forward to s'_4 (since $V_I(s'_{2,5}) > V_I(s'_4)$). The aberrational maximum acts as an attractor for nearby states, such as s'_3, and causes them to change their local policy away from optimal. An intuitive way to understand the problem is to consider a local homunculus that sits at s'_3 and can see the utilities of its neighbors. From his point of view, $s'_{2,5}$ looks desirable since once the system is in $s'_{2,5}$ it can execute a'_r, which often leads to s'_6 (one step from the goal). On the other hand, choosing the action which leads to s'_4 leaves the system still three steps from the goal. From the homunculus' point of view, going to $s'_{2,5}$ is on average better than going to s'_4. What the homunculus cannot perceive (because of perceptual aliasing) is that going from s'_3 directly to $s'_{2,5}$ always returns the real world to state s_2, which cannot reach s'_6 directly. The problem is that the homunculus cannot distinguish between s_2 and s_5, as they are both represented by $s'_{2,5}$, and he erroneously assumes that the effects of actions only depend on the current perceived state (the Markov assumption).

The aberrational maximum is also unstable because it is based on a running average of the expected returns. If, because of policy changes, s_5 is rarely visited, the aberration will disappear. Unfortunately, as soon as the policy changes back so that s_5 begins to be encountered more frequently, the aberration reappears. Thus, the system oscillates from policy to policy, unable to converge on a stable one.

The trouble with perceptual aliasing is that it prevents the decision system from learning accurate estimates of the utility and action-value functions by causing the system to average *different* values from *different* world states. The internal state $s'_{2,5}$ can never accurately represent both (or either) s_2 and s_5 since its value and action-value functions are based on averages. Intuitively an internal state, s' is a good representation if 1) every state it represents in the actual world has the same utility and 2) there is one internal action that when executed in s' maps to the optimal action in the world. This intuition about a good representation can be formalized by introducing the notion of a consistent decision and a consistent state.

Let us begin by defining three sets that are useful for discussing relationships between states and decisions in the actual and perceived decision problems. First define $SRep(s')$ to be the set of world states that for one configuration or another of the sensory-motor system map to the internal state s'. Formally, $s \in SRep(s')$ if and only if there exists a sensory-motor configuration $c \in C$ such that $\mathcal{P}(s, c) = s'$.

Similarly, define $DRep(d')$ to be the set of actual decisions (state-action pairs in the actual decision problem) that for one configuration or another of the sensory-motor system map to the internal decision $d' = (s', a')$. Formally, $d = (s, a) \in DRep(d')$ if and only if there exists $c \in C$ such that $\mathcal{P}(s, c) = s'$ and $\mathfrak{M}(a', c) = a$.

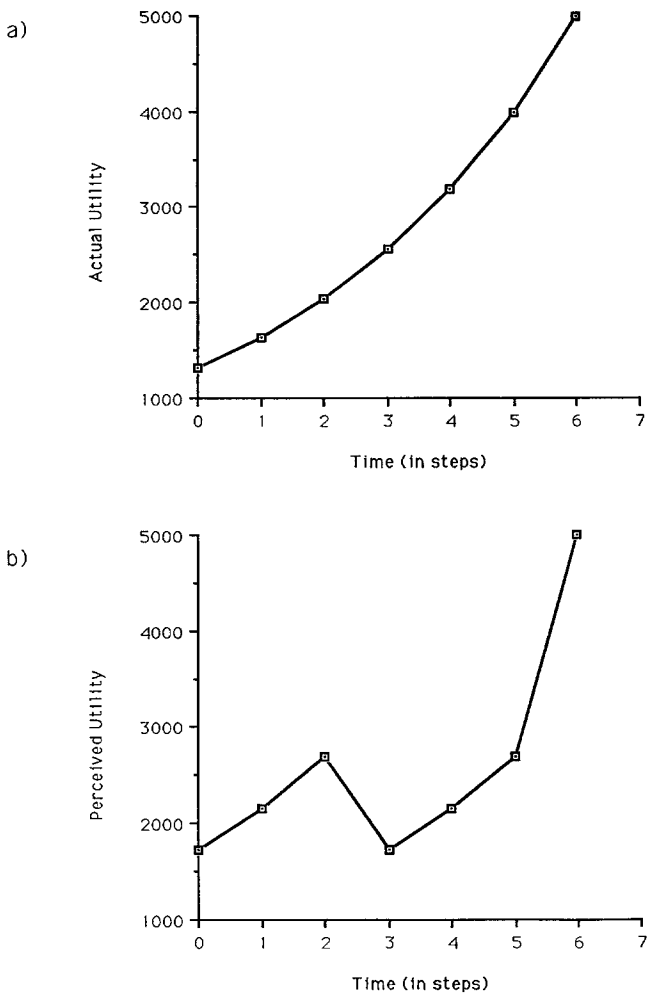


Figure 6. Plots of the utility versus time as the agent traverses from state s_0 at $t = 0$ to g at $t = 7$ (for $\gamma = 0.8$): a) the utility for the actual decision problem, V_E^*; b) the utility estimated by the decision subsystem for the perceived decision problem, V_I.

Second, the value function for the perceived problem, V_I, no longer monotonically increases as the system traverses the optimal trajectory. This anomaly is shown graphically in Figure 6b, which plots $V_I(x_t)$ as a function of time as the system follows the optimal trajectory from s_0' to g'. The figure shows that a local maximum occurs at $t = 2$ when the system first encounters $s_{2,5}'$. We call this event an *aberrational maximum* since it does not reflect the true expected return of the actual decision problem. In reality, the world is in state s_2 and the true expected return is $V_E^*(s_2)$ (= 2048 for $\gamma = 0.8$), but, because of perceptual aliasing, the decision system cannot distinguish s_2 and s_5 and, consequently,

Finally, define $Con(s, s')$ to be the set of sensory-motor configurations that map the world state s into the internal state s'. Formally, $c \in Con(s, s')$ if and only if $\mathcal{P}(s, c) = s'$.

Now, an internal decision $d' = (s', a')$ is defined to be *consistent with respect to the actual task* if and only if every decision it represents in the actual decision problem has the same optimal action-value. That is,

$$d' \text{ is } \textit{consistent} \quad \text{iff} \quad \exists_{k \in \Re} \, \forall_{d \in DRep(d')} \, [Q_E^*(d) = k]. \tag{16}$$

Similarly, an internal state s' is defined to be *consistent with respect to the actual decision problem* if the state has one decision, the optimal decision, that 1) is consistent and 2) for every world state represented by s' maps to the optimal decision in the actual decision problem. That is, s' is consistent if and only if there exists $d' = (s', a')$ such that d' is consistent and

$$\forall_{s \in SRep(s')} \, \forall_{c \in Con(s,s')} \, [\mathfrak{M}(a', c) = \pi_E^*(s))].$$

Given these definitions of consistency, internal states can be labeled as either consistent or inconsistent. Inconsistent states give rise to aberrational maxima and interfere with learning, whereas consistent states serve as true representations of the actual world state. In the example task given above, the decision $d' = (s'_{2,5}, a'_r)$ is inconsistent since the corresponding decisions in the actual problem differ (i.e., $Q_E^*(s_2, a_r) = 2048$ and $Q_E^*(s_5, a_r) = 4000$). Similarly, $s'_{2,5}$ is inconsistent since the optimal decision $(d = (s'_{2,5}, a'_r))$ is inconsistent—the optimal action-values for s_2 and s_5 differ, even though the optimal action is the same.

The negative effects of perceptual aliasing need not arise only in systems with limited perception. It can also arise when assigning credit to the internal structures of a decision system. For instance in an example similar to the one described above, Grefenstette (1988) showed how strength averaging in the rules of a classifier system, using the *bucket brigade* algorithm (Holland et al., 1986) for credit assignment, prevents the system from learning an optimal control strategy. In this case, rules that match multiple world states (allowed to improve generalization) exhibit a kind of perceptual aliasing and as a result are vulnerable to inconsistencies and inaccurate return estimates.

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 updating rule, Equation 12.

4.1. The overt cycle

The algorithm used by the new decision system is outlined in Figure 7. 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.[10]

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 internal states, S_t, for the decision

Overt Cycle:

1) Execute **Perceptual Cycle** and generate S_t, a set of internal representations for the current world state.
2) Estimate the utility of the current world state, s_t: $V_E(s_t) \leftarrow \max_{s \in S_t}[V_I(s)]$.
3) Execute **Update-Overt-Q-Estimates** based on $V_E(s_t)$, r_t, $oact_{t-1}$ and $lion_{t-1}$; where r_t is the reward received at time t, $oact_{t-1}$ is the last overt action executed, and $lion_{t-1}$ is the internal state selected to represent the previous world state (see below).
4) Choose the next overt action to execute:
   With probability p follow policy:
   $oact \leftarrow a_\pi$ such that $\exists_{s_\pi \in S_t}[Q_I(s_\pi, a_\pi) = \max_{(s,a) \in S_t \times A_O}[Q_I(s, a)]]$
   Otherwise choose randomly: $oact \leftarrow R(A_O)$
5) Select the *Lion*: $Lion \leftarrow (s_L, oact)$ such that $Q_I(s_L, oact) = \max_{s \in S_t}[Q_I(s, oact)]$.
6) Execute $oact$ to obtain a new world state and go to 1).

Update-Overt-Q-Estimates:

1) Estimate the error in the lion's action-value: $error \leftarrow (r_t + \gamma V_E(s_t)) - Q_I(Lion_{t-1})$.
2) Update the action-value of the *Lion*:
   If $(error < 0)$ then the lion is suspected of being inconsistent, so suppress it: $Q_I(Lion_{t-1}) \leftarrow 0.0$
   Else update it using the standard 1-step Q-learning rule: $Q_I(Lion_{t-1}) \leftarrow Q_I(Lion_{t-1}) + \alpha error$.
3) Update non-lion internal states:
   For each $s \in S_{t-1}$ and $s \neq state(Lion_{t-1})$ do:
   $Q_I(s, oact_{t-1}) \leftarrow Q_I(s, oact_{t-1}) + \alpha' error$ — where $\alpha' < \alpha$.

Perceptual Cycle:

1) Initialize S_t: $S_t \leftarrow \{s_c\}$, where s_c is the current internal state.
2) Do n times: (in our implementation $n = 4$)
   i) select the next perceptual action:
   with probability p' : $pact \leftarrow a$ such that $Q_I(s_c, a) = \max_{b \in A_P}[Q_I(s_c, b)]$,
   otherwise: $pact \leftarrow R(A_P)$.
   ii) execute $pact$ to obtain a new internal state s'.
   iii) update the action-value estimate for the decision $(s_c, pact)$:
   $Q_I(s_c, pact) \leftarrow Q_I(s_c, pact) + \alpha(V_I(s') - Q_I(s_c, pact))$.
   iv) add s' to S_t: $S_t = S_t \cup \{s'\}$
   v) update s_c: $s_c \leftarrow s'$.
3) Return S_t.

Figure 7. An outline of the steps executed by the new decision system designed specifically to overcome the difficulties caused by perceptual aliasing.

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

$$Q_I(s_\pi, a_\pi) = \max_{(s,a) \in S_t \times A_O} [Q_I(s, a)]. \tag{17}$$

Once an overt action is chosen, it is executed and the overt cycle begins anew. Figure 8 shows a cartoon of the new decision system in action. The large nodes represent world

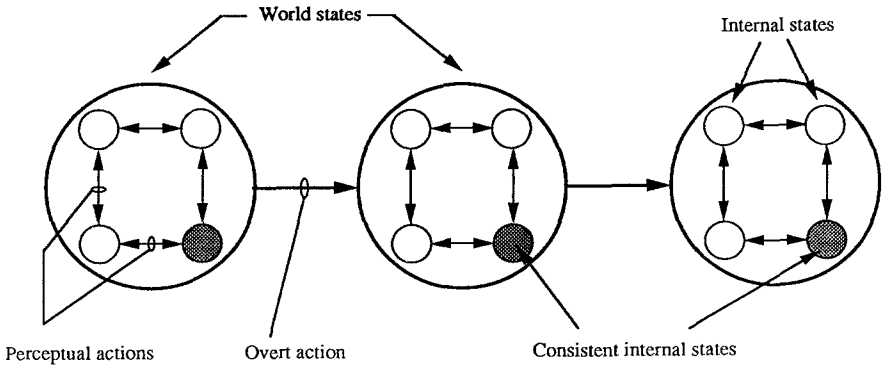


Figure 8. A sketch of the new decision system in action. The large (super) graph depicts the overt cycle, where large nodes correspond to world states and arcs correspond to overt actions. The subgraphs embedded within each large node depict perceptual cycles, with nodes corresponding to internal representations of the current world state and arcs corresponding to perceptual actions.

states, and the arcs between them overt actions. Embedded within each large node is a subgraph representing the perceptual cycle. The nodes in this graph correspond to representations seen by the decision system, and the arcs between them correspond to perceptual actions.

4.2. 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 (cf. Equation 5). 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,

$$Lion = (s_L, a_L) \quad \text{such that} \quad Q_I(s_L, a_L) = \max_{s \in S_t}(Q_I(s, a_L)). \tag{18}$$

When the system is following its policy, the lion is just the policy decision, $Lion = \mathrm{d}_\pi$.

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:

$$Q_I^{ideal}(s, a) = \begin{cases} Q_I^*(s, a) & \text{if } (s, a) \text{ is consistent} \\ 0 & \text{otherwise.} \end{cases} \quad (19)$$

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. The steps for updating action-values are shown in Figure 7 under the **Update-Overt-Q-Estimates** heading.

4.3. The perceptual subcycle

The steps in the perceptual cycle are sketched in Figure 7 under the **Perceptual Cycle** heading. 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.[11] 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, as shown in Figure 7 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 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 (cf. Section 5). 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 once 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.

5. An example

To test our ideas, we implemented a system that learns a simple block manipulation task. In this task, the agent is presented with a pile of blocks on a conveyor belt. The agent can manipulate the pile by picking and placing blocks. When the agent arranges the blocks in certain *goal* configurations, it receives a fixed reward of 5000 units. Otherwise, it receives no reward. When the agent solves the puzzle, the pile immediately disappears and a new pile comes down the belt. If the agent fails to solve the puzzle after a fixed number of steps, n_{quit}, the pile falls off the end of the conveyor and a new pile appears at the front. A pile can have any number of blocks in it and can be arranged in arbitrary stacks. A

block can be any one of three colors: red, green, or blue. We make the standard assumptions that a block can be picked up only if its top is clear and that a block can be placed only at locations with clear tops.

We have implemented a program that learns to solve a simple version of the block manipulation game. The program uses the deictic sensory-motor system described in Figure 3, and the decision system described above. The objective of the program is to demonstrate the feasibility of the lion algorithm and no special effort has been taken to optimize performance.

The particular task we studied rewards the agent whenever it picks up a green block. That is, goal configurations consist of those states in which the robot is holding a green block. We chose to study this task because it is very simple, but adequate to demonstrate the difficulties caused by perceptual aliasing. These problems can be seen in Figure 9, which shows a typical sequence of world states the agent might traverse in solving one instance of the problem. The trick to this task is for the agent to learn to focus on the green block. That is, depending upon the placement of the attention frame, world states 1,3,4,5, and 6 may have inconsistent internal representations. If the attention frame is fixed on the green block, then the internal states are consistent; if the attention frame is fixed on any other block, then the internal representations of the states are inconsistent. For example, in state 6, if the attention frame is fixed on the blue block, then state 6 cannot be distinguished from other world states that are identical except with additional blocks above the green block. The system overcomes this ambiguity by learning to direct its attention frame to the green block, which provides sufficient extra information (the height of the green stack) needed to disambiguate the situation.

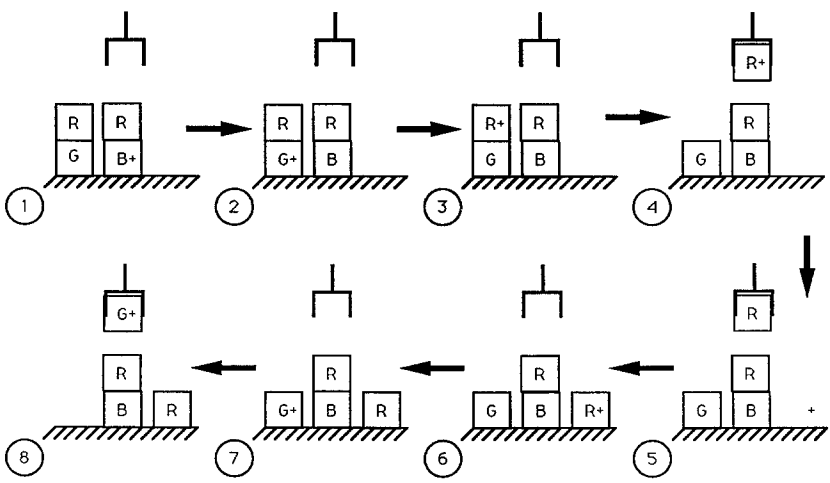


Figure 9. A sequence of world states in a typical solution path for the block manipulation task. Depending upon the placement of the attention frame, states 1, 3, 4, 5, and 6 may be represented ambiguously. The (+) shows the course of the action frame. Not shown are the states of the perceptual subcycle, where the attention frame moves.

A series of experimental runs were performed to obtain qualitative data on the new decision system's performance. In each run, the robot was sequentially presented with 1000 instances of the task (i.e., 1000 trials). Each instance consisted of a randomly configured pile of 4 blocks, with the pile always containing exactly one green block. Randomly selecting problem instances guaranteed that the system would get a good mix of easy and difficult problems. An easy problem, for instance, is one in which all three blocks are uniformly placed on the table. In this case, the robot need merely fixate and grasp the green block. Conversely, a more difficult problem is one in which the green block is at the base of a stack containing 3 blocks. In this case, the robot must unstack each block until it clears the green one. If in any trial the robot fails to solve the problem after n_{quit} overt actions, it decides that the instance is too difficult and moves on to the next trial.

Performance results for a typical experimental run are shown in Figure 10. The graph shows the number of overt actions taken by the agent for each of the 1000 instances of the task it encounters during a typical run. Initially, the agent fails on almost every trial (i.e., it takes 30 steps and quits). It does, however, manage to solve a few instances. These early successes are invariably easy problems, requiring only one or two correction actions to solve. After about 100 trials, the agent begins to solve more and more instances including more difficult problems. Eventually, the agent learns to solve even the most difficult instances and rarely fails (e.g., $< 5\%$ failure after 1000 trials).

The agent's performance on a given trial depends strongly on the difficulty of the trial instance; consequently, Figure 10 appears very noisy. A clearer picture of the agent's performance is obtained by averaging results over multiple experimental runs. Figure 11 plots the average over 200 runs of the number of overt actions taken per trial. Plots for the optimal

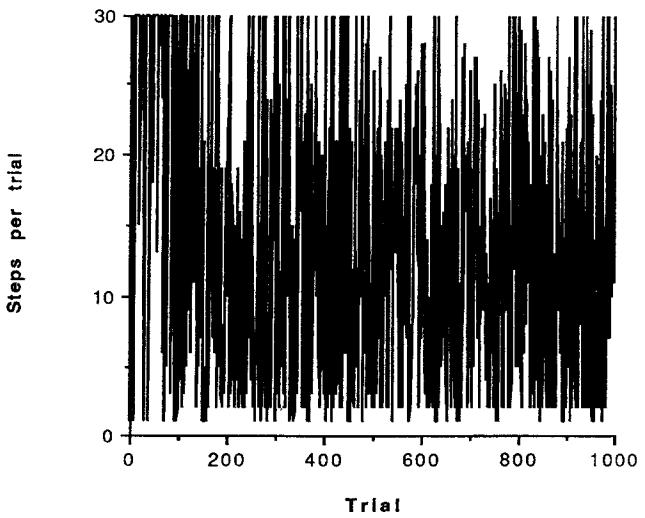


Figure 10. A plot of the number of steps per trial as a function of the instances seen by the agent for a typical experimental run. Noisiness is due to the wide variety of tasks being solved.

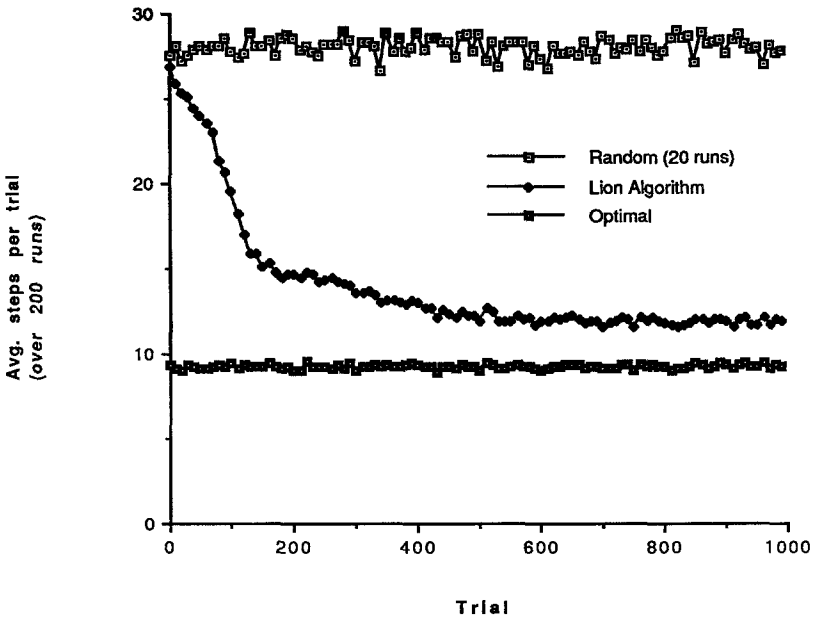


Figure 11. A plot of the average number of steps per trial as a function of the instances seen by the agent. The average is taken over 200 runs and provides a smoother picture of the agent's learning curve. The agent's steady state performance is approximately 125% optimal. Also shown is the average number of steps taken by an agent acting randomly.

number of steps (average of 200 runs) and for an agent behaving randomly are also shown. The figure clearly shows that the agent's initial performance is poor—near the maximum of 30 steps per trial—but improves considerably during the first few hundred trials. The agent's performance settles just under 12 steps per trial (about 125% optimal).

The agent's performance fails to converge to optimal for two reasons. First, with probability $1-p$ ($p = 0.9$), the decision subsystem chooses its overt action randomly, reflecting a simplification in our decision algorithm that can be eliminated by incorporating a slightly more complex procedure for controlling exploration (Barto et al., 1990b). Second, the decision subsystem is not guaranteed in every case to find a consistent lion (even if it exists) since in our implementation the perceptual subcycle only executes 4 perceptual actions and chooses the lion from the set of at most five unique internal states. Further, perceptual actions are also occasionally ($p' = 0.9$) selected randomly. As a result, residual inconsistent lions occasionally arise and interfere with the agent's performance.

Figures 10 and 11 show that the agent learns to solve the task, but they say nothing about which instances the agent learns to solve first or the order in which the agent learns its task-dependent representation. To get a glimpse of the order in which the instances of the task are learned, each problem instance was classified into one of four categories: easy, intermediate, difficult, and very difficult. Easy problems correspond to instances in which the green block is clear and the agent need only pick it up. Intermediate problems include

instances where the green block is covered by one block; difficult problems, two blocks; and very difficult problems, three blocks. Plots of the average trial times and average success rate for each of these four classes of problems are shown in Figure 12 and Figure 13, respectively. Both figures show that the agent first learns to solve easy tasks reliably, and then learns more and more difficult ones. In Figure 12, the agent shows improvement on easy tasks immediately; it shows improvement on intermediate tasks after 10-20 trials; on difficult tasks after 50-60 trials; and on the most difficult tasks after 70-80 trials (see Figure 12b). A similar trend is seen in Figure 13, which also shows that the agent eventually learns to reliably solve all but the most difficult tasks and then only fails about 10% of the time.[12]

To determine the order in which the agent learns a consistent representation, statistics were collected to measure the amount of overestimation that occurs during learning. As before, world states were classified into four categories according to their distance to the goal: easy, intermediate, difficult, and most difficult. For each class, the fraction of times (over 200 experimental runs) the lion overestimated (and was suppressed) was maintained as a function of the number of trials seen. These percentages are plotted in Figure 14. As expected, the agent initially overestimates a high fraction of the time. This fraction is especially high because a single overestimation can cause a chain of subsequent overestimations; and lacking knowledge on how to control perception, the agent frequently fails to choose a consistent lion. With experience, however, the agent eventually learns to select consistent internal states, and the amount of overestimation decreases.

We expected the agent to learn consistent lions for easy states first and then to boot-strap its way to consistency for more and more distal states. To some extent this expectation is verified in Figure 14, which shows that the amount of overestimation decreases first for easy states and decreases later for more difficult states. Early on, the fractions for intermediate, difficult, and most difficult problems are rarely solved, and when they are, they tend to be inefficient. For example, when solving an intermediate problem, it is common for the agent to stack an extra block on the green pile, try other unhelpful actions within that configuration for a while, unstack the block, and go on to solve the problem. Thus, the agent sees mixes of intermediate, difficult, and most difficult states. Initially, therefore, all trials end up visiting about the same fraction of consistent states. This random searching is much less prevalent in easy tasks whose solutions involve only one or two correct actions. Eventually, as the agent learns to solve easy problems (after 80-100 trials), intermediate states become increasingly consistent and the agent visits harder states less frequently on its way to the goal. The inconsistency in intermediate states tends to decrease while the consistency of more difficult states remains unchanged.

Figure 14 also shows that after 1000 trials the agent continues to overestimate a substantial fraction of the time. This fraction is fairly low for easy states ($\approx$ 5%) but unexpectedly high for the most difficult states ($\approx$ 45%). There are three reasons for this high rate of overestimation. First, as previously mentioned, our implementation is not guaranteed to always find a consistent internal state, even if one exists. This explains the small fraction of steady state overestimation that occurs even for easy states. Second, a single overestimation (and suppression) tends to cause a chain reaction of overestimations in earlier "set-up" states (even for consistent states). Thus, the high fraction of overestimation in more distal (difficult) states is explained by the fact that occasional overestimations in easy states propagate

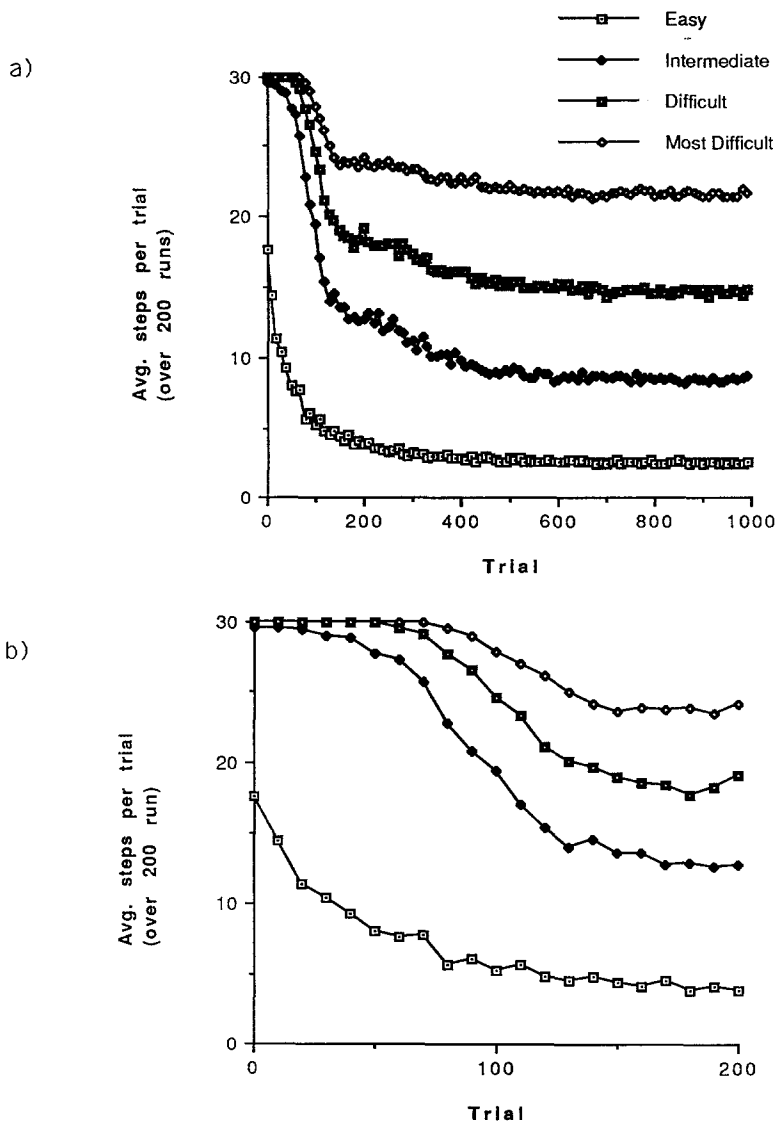


Figure 12. Plots of the average number of steps per trial for each of the four classes of problem instances: i) easy (no unstacking); ii) intermediate (1 to unstack); iii) difficult (2 to unstack); and iv) most difficult (3 to unstack). a) shows a complete plot ranging from 0 to 1000 trials; b) shows a focused plot ranging from 0 to 200 trials. The plots show that the agent learns to solve easier tasks first.

back to these states and destroy consistencies there. Third, when overestimations occur they tend to temporarily break the agent's decision policy. Often the agent will waste a great deal of time in an inconsistent confused loop until it gives up or manages to luck

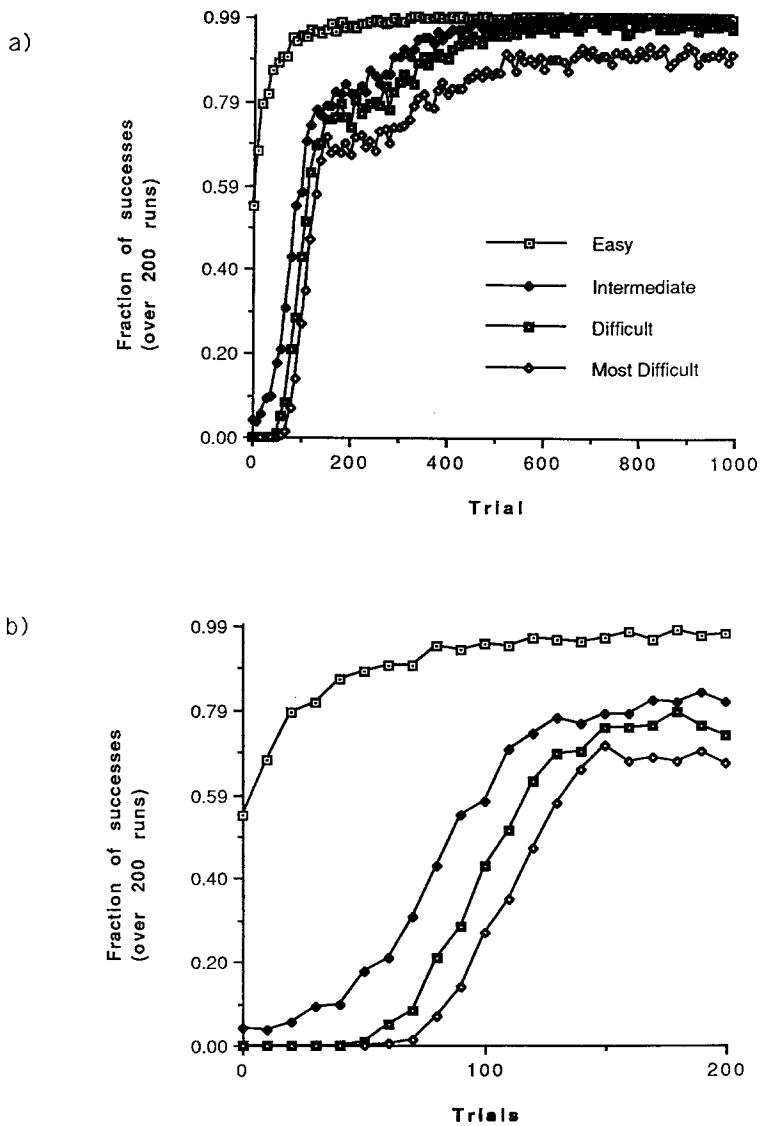


Figure 13. Success rates for each of the four classes of problem instances versus the number of trials seen by the agent. a) shows a complete plot ranging from 0 to 1000 trials; b) shows a focused plot ranging from 0 to 200 trials. The plots show that the agent learns to solve easier tasks first and eventually learns to solve all instances fairly reliably.

into a state from which it can solve the problem. As a result, these statistics are misleading in that they tend to report more overestimations than there are inconsistent internal states.

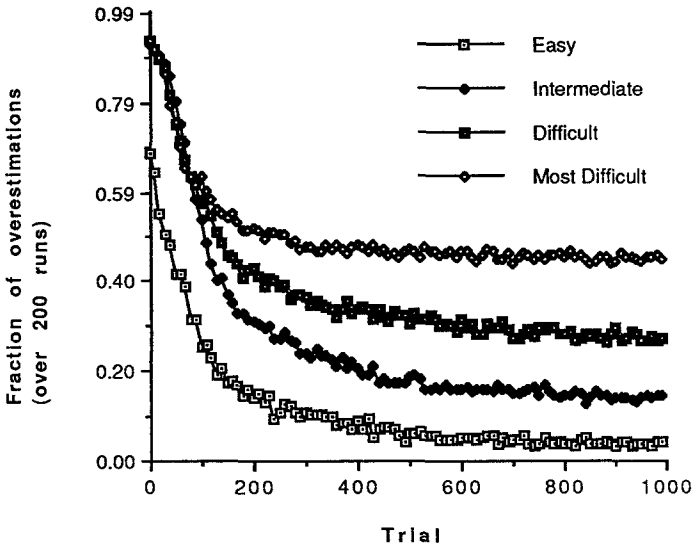


Figure 14. The fraction of overestimations encountered over 200 runs for each of the four classes of world states. The plot shows that consistent representations are learned for easy states first, followed by consistent representations for more difficult states, and that the agent continues to perform in the face of residual inconsistencies and overestimation.

The robustness of the agent's performance in the face of persistent overestimations led us to consider tasks with more than four blocks. Another set of experiments were performed in which the problem instances ranged from easy (0 blocks to unstack) to most difficult (3 blocks to unstack). In these experiments, however, additional outlying blocks were added to the pile. The number of outliers was randomly chosen between 0 and 20. Outliers interfere with the system's ability to learn the most difficult instances because the agent's sensory motor system cannot distinguish between stacks containing four or more blocks. Therefore, the agent has no way of distinguishing (under any sensory-motor configuration) states where it has to unstack three blocks from states where it has to unstack 4,5,6, or more blocks. These states do not have consistent internal representations. Results from the experiments are shown in Figure 15. They are comparable to the results from our earlier experiments, except with slightly longer average solution times and a slightly lower success rate (especially for the most difficult instances). Nevertheless, even in the face of inconsistencies the agent is capable of learning a robust decision policy.

6. Discussion: Limitations and future prospects

In this section current limitations of the architecture and the lion algorithm are discussed. Where possible, we also outline approaches that may be useful in overcoming these limitations. As yet, none of the suggestions outlined have been implemented or tested; therefore, the discussion is necessarily speculative.

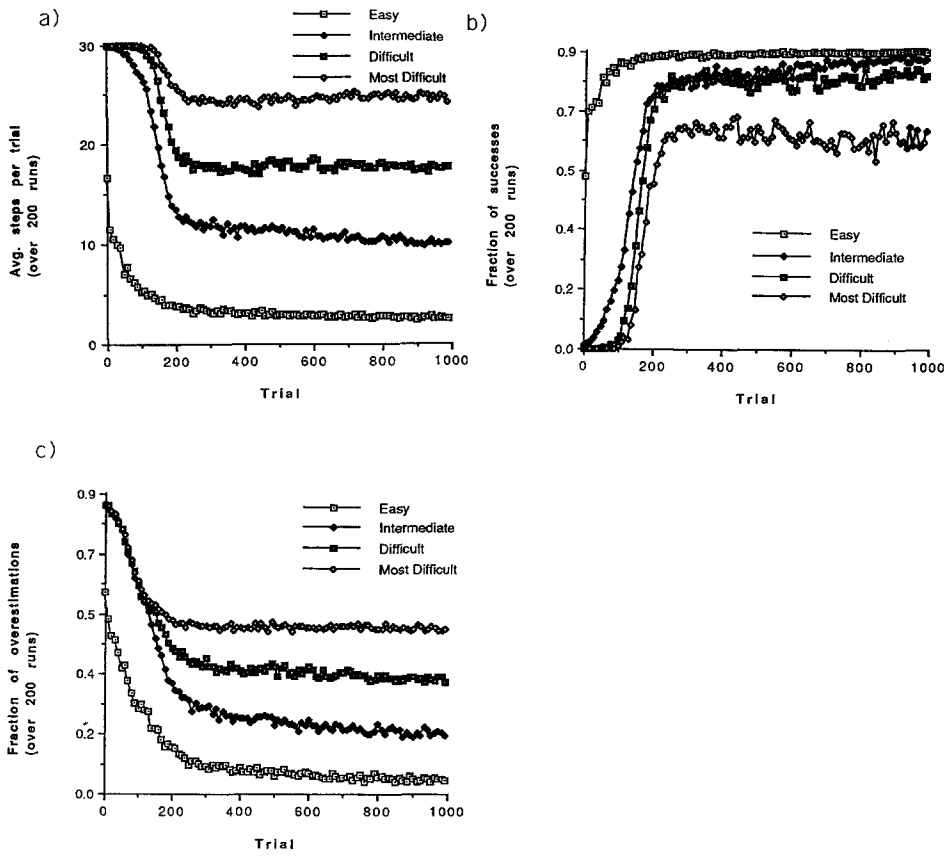


Figure 15. Performance plots for experiments that include piles of up to 20 outlying blocks. a) The average solution time for each class of problem, b) the success rate for each class of problem, c) the fraction of overestimations observed for each class of state. The plots are comparable to those in our original experiments and show that the agent can learn even in environments which it cannot consistently represent.

6.1. Deterministic tasks

One of the most important assumptions made in our model is that the external world is deterministic. The lion algorithm depends upon the world being deterministic to differentiate consistent representations from inconsistent ones. That is, internal states whose optimal returns have non-zero variance (detected by overestimation) are inconsistent and are suppressed. If the world were allowed to be non-deterministic (i.e., stochastic), then the optimal returns of consistent internal states would also have a non-zero variance and the lion algorithm would weed them out as well.

At the moment, we do not know how to deal in general with stochastic worlds (e.g.,

when the world is modeled by a Markov decision process.) However, we believe that minor modifications to the lion algorithm can lead to systems that can cope with two restricted (but useful) classes of non-determinism.

Quasi-Deterministic Transitions: Imagine a world which is more or less deterministic except that occasionally, due to unperceivable circumstances, a random perturbation occurs. This perturbation might cause the world to make an "unexpected" transition (e.g., the stack of blocks tips over), or cause the agent to receive an anomalous reward (e.g., a food pellet gets wedged into the injection mechanism). The current lion algorithm is extremely sensitive to such "failures" and would suppress even consistent lions (good internal representations) upon encountering even one failure. This suppression can have a catastrophic impact on the stability of the optimal policy because it can lead to a chain reaction in which a whole series of consistent lions gets suppressed. Thus, one untimely failure can completely destroy an otherwise perfect policy. One way to overcome this limitation is to allow lions to occasionally overestimate, thereby allowing occasional failures without severely impacting the stability of the optimal policy. Inconsistent lions would still tend to be suppressed since they act as attractors and cause the agent to repeatedly visit (and overestimate) them.

Minor Payoff Variation: Another source of non-determinism might be caused by slight variations in the reward function. For example, upon successfully traversing a maze, a (artificial) rat might receive food pellets that vary slightly in size (and reward value). The current lion algorithm is sensitive to this noise and would have trouble learning the task. One approach to this problem is to permit a certain amount of overestimation. That is, instead of suppressing a lion whenever it overestimates its return (i.e., $error \times 0$), we suppress it only if it overestimates by too much (e.g., $error \times threshold$).

Another alternative is to suppress the lion only partially. That is, instead of resetting overestimating lions to zero, the action-value is reset to some fraction (say 95%) of the lower estimate. In this case, in the face of noisy returns, consistent states would tend to take on values near their average (i.e., roughly their true consistent values), while inconsistent states would tend to take on values around a fraction (95%) of their lowest values.

6.2. Non-blocked tasks

Instances in a problem-solving task are presented in blocks of trials. Once the agent solves a problem instance, the trial ends and a new instance begins. Defining tasks in terms of trials and distinguished goal states provides the learning algorithm with the foothold it needs to learn its *first* problem (and consistent internal states), which is used in turn to boot-strap to more and more difficult instances. At the moment, it is not clear if the current algorithm will work for less structured tasks that continue indefinitely instead of ending once the agent receives a reward.

6.3. Adequate perception

Limited Sensors: Another major assumption made by the lion algorithm is that each world state is consistently represented by at least one internal state. This assumption becomes

problematic for tasks that require a large amount of information to uniquely describe the absolute state of the world with respect to the task. The difficulty can be seen by considering the fruitcake (or block copying) problem described by Ginsberg (1989). In the fruitcake problem, the agent's goal is to arrange a stack of lettered blocks so that the stack spells out the work "fruitcake". Chapman has shown that a system with a deictic sensory-motor system (which uses 4 markers) can be *built* to solve the problem. But, our current system, even if equipped with the additional 4 markers is unable to *learn* this problem because it cannot unambiguously encode the state of the world with just four markers. If each marker reports only the letter and position of the block it marks, then placing the agent's markers on any four blocks in the stack spelling fruitcake does not provide enough information for the agent to know that it has correctly spelled "fruitcake." At best it can know that four letters are in their correct position. Adding additional markers, one for each letter in the word, would help but is not satisfactory because Chapman has shown that a system with four markers is sufficient.

Loops: A similar problem arises for tasks whose strategies require the agent to repeatedly execute a loop a large number of times. For example, in the block manipulation task we studied, the agent could learn to solve only instances where 4 or fewer blocks were above the green block (although many more blocks could be in the pile), because its sensory system had only 2 bits to encode the stack height.

One approach to this problem is to consider grouping sets of decisions into *macro* decisions (or schemas) that capture the fundamental cycles in the policy and treat them as unit actions.

Memoryless tasks: Another closely related assumption made by the architecture is that the agent's local environment completely encodes the state of the task. That is, if equipped with a sufficiently complex sensory system, the agent can consistently represent the world by sensing its local environment. This assumption prohibits the agent from learning tasks that require short-term memory. During the course of a task the agent may receive a signal that determines a decision to be made in the future. If the agent has no way to remember the value of the signal, it will not be able to make the correct decision when the time comes. For example, a bus driver, upon hearing the "exit bell," must remember the signal long enough to reach the next bus stop.

It would be interesting to consider architectures that incorporate mechanisms for memory and recall. Presumably, memories could be stored and recalled based on the indexical aspects that make them significant (i.e., like perception).

6.4. Faster learning

In this article, we have chosen simplicity over speed. However, the rate at which these systems learn is an important research issue deserving further attention. Four approaches to improving the learning rate are given below. It would be interesting to see how well they can be integrated into our existing architecture.

Generalizing Function Approximators: Our current experimental system uses a table to implement the action-value function and therefore is incapable of generalizing over states (although generalization occurs implicitly through the use of a deictic sensory-motor system).

One way to speed up learning is to use more sophisticated function approximation techniques that are capable of generalization (e.g., CMACs (Albus, 1975; Albus, 1981), neural networks (Anderson, 1986), rule based classifiers (Holland, 1975; Holland et al., 1986), and clustering algorithms (Mahadevan & Connell, 1990)).

Explanation-based generalization: A related approach to generalization has recently been suggested by Yee et al. (1990) and involves using explanation based generalization (EBG) to learn useful concepts based on the system's ability to explain expectation failures. This approach requires that the agent possess a formal model of the world, which it uses to form explanations and to perform regression. The technique allows credit to be assigned not only to states that immediately precede a particular reward event, but also, through a generalized explanation of the causal sequence, to states that are functionally equivalent. Gordan and Grefenstette (1990) have also considered combining EBL and reinforcement learning techniques.

Forward modeling: A third approach to improving the learning rate is for the agent to learn a forward model of the world and simultaneously use that model to perform mental experiments. This approach has recently been shown to significantly improve the learning rate for simple (standard) reinforcement learning tasks (Sutton, 1990a; Sutton, 1990b; Whitehead & Ballard, 1989b; Whitehead, 1989). It would be interesting to see how well these ideas can be applied to architectures with active sensory-motor systems.

Supervision: Finally, to obtain very fast learning, it will ultimately become necessary to learn from a teacher (Whitehead & Ballard, 1991). As a first step in this direction, it would be useful to study the stability of systems that first receive immediate feedback (e.g., hot/cold information) from an external teacher and eventually learn to solve the task without intervention. Although this approach is viable for standard reinforcement learning algorithms, it appears to be problematic for the lion algorithm since removing the teacher's rewards would cause the system to overestimate its returns. Thus, the current lion algorithm would unlearn (via suppression) everything it gained by the teacher. The modified lion algorithm that resets overestimating lions to 95% of the lower value should overcome this problem and be amenable to instruction.

7. Conclusions

In this article, we have considered the interactions that arise in adaptive control architectures that integrate active sensory-motor systems (specifically deictic representations) with decision systems based on reinforcement learning. We found the integration non-trivial because active sensory-motor systems naturally lead to internal states that are inconsistent. Inconsistent states wreak havoc on the decision system's ability to learn by introducing apparitional maxima in the value function and destabilizing the learning algorithm with respect to the optimal policy. A solution to this problem, based on actively detecting and suppressing inconsistent states, was proposed. The result is a system that learns to focus its attention on the relevant aspects of the domain as well as control its overt behavior. The new algorithm was demonstrated in a system that learns a simple block manipulation task that is beyond the scope of previous reinforcement learning systems. Although our systems are still very primitive, we find the results encouraging and hope that continued effort will yield systems capable of more sophisticated behavior.

Acknowledgments

We are especially thankful to Josh Tenenberg for his insightful discussions during the formative stages of this work. We also gratefully acknowledge Phil Agre, Chuck Anderson, Andrew Barto, Chris Brown, Steve Bradtke, Bulent Murtezaoglu, Ray Rimey, Rich Sutton, Paul Viola, and Lambert Wixson for their technical feedback, and thank Kathy Ivey for thoroughly proofreading a draft version of this paper. We also thank the anonymous reviewers for their feedback. The comments of one reviewer, in particular, were especially detailed and helpful.

Notes

1. In principle, the world could include the agent in its description. However, for the simple tasks we are concerned with, it suffices to view the world simply as a model of the agent's local external environment.
2. In this article we have included the reward function as part of the world model. This is consistent with the definition of a Markov decision process and conveniently separates the agent from the task it is trying to learn. In terms of a physical realization, the reward function could be implemented in a reward center of the agent's nervous system. In this case, the agent encodes a priori knowledge about the task since the reward center knows when to generate a reward. In such a configuration, however, the reward center would tend to be relatively simple, and rewards would be few and far between. For example, in a problem-solving task, the reward center might generate a non-zero reward only when the agent achieves the goal, but not otherwise. Or, alternatively, the reward center might act as a go-between, generating a reward only upon receiving specific stimulation from an external source (e.g., a teacher).
3. Our definition of V_{π_E} varies slightly from the definition commonly found in dynamic programming texts. This variation is an artifact of our decision to associate rewards with states instead of associating them with the choice of an action in a state.
4. Even though problem-solving tasks, as we've defined them, can be studied without introducing the extra notation associated with Markov Decision Problems, we have adopted the more general framework because 1) it is commonly used to model sequential decision processes both in machine learning and elsewhere, 2) it adds perspective to the class of tasks we are studying, and 3) it serves as a goal, in that we hope eventually to extend our results to this more general class of problems.
5. In general, it is possible to use updating rules based on a weighted sum of n-step returns. Sutton (1988) takes this approach in his Theory of Temporal Differences (TD) Methods. Two advantages of TD-methods are 1) $Q_t(x_t, a_t)$ is updated after each step (and thus retains the advantages of both small and large n) and 2) TD-methods have efficient implementations since, by choosing the weights just right, the estimation error can be based on the differences between the predictions of temporally adjacent states.
6. **deic·tic** \ dī-ik-tik, dā-; dē-ik- \ *adj* [Gk *deiktikos*, fr. *deiktos*, verbal of *deiknynai* to show] : showing or pointing out directly ⟨the words *this*, *that*, and *those* have a ~ function⟩ (from *Webster's New Collegiate Dictionary*, ninth edition)
7. The term *marker* originated in Ullman's work on visual routines (Ullman, 1984).
8. In this discussion, we finesse the issue of exactly what an "object" is and assume an object can be defined by its local features, such as its shape and color.
9. Agre and Chapman do not distinguish markers as overt or perceptual in their systems. The two classes are introduced here because the learning algorithm described below depends on being able to distinguish between actions that change the world and actions that simply change the agent's perception of it.
10. As a side effect, overt actions may also change the perceptual configuration, but perceptual actions are not allowed to affect the world state. In the deictic sensory-motor system described in Figure 3, the action-frame has only overt actions, and the attention-frame has only perceptual actions.
11. 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.

12. Increasing n_{quit} slightly, say to 40, almost always gives the agent the extra time it needs to solve even the most difficult problems.

References

Agre, P. E. (1988). *The dynamic structure of everyday life*. PhD thesis, MIT Artificial Intelligence Laboratory, Cambridge, MA.

Agre, P. E., & Chapman, D. (1987). Pengi: An implementation of a theory of activity. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 268–272). Los Altos, CA: Morgan Kaufmann.

Albus, J. S. (1975). A new approach to manipulator control: Cerebellar model articulation controller (CMAC). *Transactions of the ASME: Journal of Dynamic Systems, Measurement and Control, 10*, 25–61.

Albus, J. S. (1981). *Brains, behavior, and robotics*. Peterborough, NH: BYTE Books.

Anderson, C. W. (1986). *Learning and problem solving with multilayer connectionist systems*. PhD thesis, University of Massachusetts, Amherst, MA.

Anderson, C. W. (1989). Towers of hanoi with connectionist networks: Learning new features. *Proceedings of the Sixth International Conference on Machine Learning* (pp. 345–350). San Mateo, CA: Morgan Kaufmann.

Ballard, D. H. (1989). Reference frames for animate vision. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 1635–1641). Los Altos, CA: Morgan Kaufmann.

Barto, A. B., & Sutton, R. S. (1981). Landmark learning: An illustration of associative search. *Biological Cybernetics, 42*, 1–8.

Barto, A. B., Sutton, R. S., & Watkins, C. (1990a). Sequential decision problems and neural networks. In D. S. Touretzky (Ed.), *Advances in neural information processing systems 2*. San Mateo, CA: Morgan Kaufmann.

Barto, A. B., Sutton, R. S., & Watkins, C. J. C. (1990b). Learning and sequential decision making. In M. Gabrial & J. W. Moore (Eds.), *Learning and computational neuroscience*. Cambridge, MA: MIT Press. (Also COINS Tech Report 89–95, Dept. of Computer and Information Sciences, University of Massachusetts, Amherst, MA 01003).

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuron-like elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-13*, 834–846.

Bellman, R. E. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.

Bertsekas, D. P. (1987). *Dynamic programming: Deterministic and stochastic models*. Englewood Cliffs, NJ: Prentice-Hall.

Blum, L., & Blum, M. (1975). Toward a mathematical theory of inductive inference. *Information and Control, 28*, 125–155.

Blythe, J., & Mitchell, T. M. (1989). On becoming reactive. *Proceedings of the Sixth International Conference on Machine Learning* (pp. 255–259). San Mateo, CA: Morgan Kaufmann.

Booker, L. B. (1982). *Intelligent behavior as an adaptation to the task environment*. PhD thesis, University of Michigan.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation, 2*, 14–22.

Chapman, D. (1989). Penguins can make cake. *AI Magazine, 10*, 45–50.

Clocksin, W. F., & Moore, A. W. (1988). *Some experiments in adaptive state-space robotics*. (Technical report). University of Cambridge, Computer Laboratory.

Drummond, M. (1989). Situated control rules. *Proceedings of the Rochester Planning Workshop* (pp. 18–34). (Technical Report 284). University of Rochester, Department of Computer Science.

Fikes, R. E., Hart, P. E., & Nilsson, N.J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence, 3*, 251–288.

Firby, R. J. (1987). An investigation into reactive planning in complex domains. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 202–206). Los Altos, CA: Morgan Kaufmann.

Franklin, J. A. (1988). Refinement of robot motor skills through reinforcement learning. *Proceedings of the 27th IEEE Conference on Decision and Control*. Austin, TX.

Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 677–682.). Los Altos, CA: Morgan Kaufmann.

Gibson, J. J. (1979). *The ecological approach to visual perception*. Boston, MA: Houghton Mifflin.

Ginsberg, M. L. (1989). Universal planning: An (almost) universally bad idea. *AI Magazine, 10*, 41–44.

Girosi, F., & Poggio, T. (1989). *Networks and the best approximation property* (AI Memo No. 1164). Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

Gordon, D. G., & Grefenstette, J. J. (1990). Explanations of empirically derived reactive plans. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 198–203). San Mateo, CA: Morgan Kaufmann.

Grefenstette, J. J., Ramsey, C., & Schultz, A. (1990). Learning sequential decision rules using simulation and competition. *Machine Learning, 5*, 355–382.

Grefenstette, J. J. (1988). Credit assignment in rule discovery systems based on genetic algorithms. *Machine Learning, 3*, 225–245.

Grefenstette, J. J. (1989). Incremental learning of control strategies with genetic algorithms. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 340–344). San Mateo, CA: Morgan Kaufmann.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Holland, J. H. (1986). Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach (Volume II)*. San Mateo, CA: Morgan Kaufmann.

Holland, J. H., Holyoak, K. F., Nisbett, R. E., & Thagard, P. R. (1986). *Induction: processes of inference, learning, and discovery*. Cambridge, MA: MIT Press.

Hormel, M. (1989). A self-organizing associative memory system for control applications. In D. S. Touretzky (Ed.), *Advances in neural information processing systems 1*. San Mateo, CA: Morgan Kaufmann.

Kaelbling, L. P. (1989). A formal framework for learning in embedded systems. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 350–353). San Mateo, CA: Morgan Kaufmann.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning, 1*, 11–46

Mahadevan, S., & Connell, J. (1990). *Automatic programming of behavior-based robots using reinforcement learning* (Research Report RC 16359). IBM T. J. Watson Research Center.

Miller, W. T., Sutton, R. S., & Werbos, P. J. (1990). *Neural networks for control*. Cambridge, MA: MIT Press.

Nilsson, N. J. (1989). Action networks. *Proceedings of the Rochester Planning Workshop* (Technical Report 284) (pp. 36–68). University of Rochester, Department of Computer Science.

Ramsey, C., Schultz, A., & Grefenstette, J. (1990). Simulation-assisted learning by competition: Effects of noise differences between training model and target environment. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 211–215). San Mateo, CA: Morgan Kaufmann.

Ross, S. (1983). *Introduction to stochastic dynamic programming*. New York, NY: Academic Press.

Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable domains. *Proceedings of Ninth International Joint Conference on Artificial Intelligence* (pp. 1039–1046). Los Altos, CA: Morgan Kaufmann.

Schoppers, M. J. (1989). *Representation and automatic synthesis of reaction plans*. PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts at Amherst.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning, 3*, 9–44.

Sutton, R. S. (1990a). First results with DYNA, an integrated architecture for learning, planning, and reacting. *Proceedings of the AAAI Spring Symposium on Planning in Uncertain, Unpredictable, or Changing Environments*.

Sutton, R. S. (1990b). Integrating architectures for learning, planning and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216–224). San Mateo, CA: Morgan Kaufmann.

Ullman, S. (1984). Visual routines. *Cognition, 18*, 97–159. (Also in: *Visual cognition*, S. Pinker (Ed.), 1985).

Watkins, C. (1989). *Learning from delayed rewards*. PhD thesis, Cambridge University.

Whitehead, S. D. (1989). *Scaling in reinforcement learning* (Technical Report TR 304). University of Rochester, Department of Computer Science.

Whitehead, S. D., & Ballard, D. H. (1989a). Reactive behavior, learning, and anticipation. *Proceedings of the NASA Conference on Space Telerobotics* (pp. 333–344). Pasadena, CA: Jet Propulsions Laboratory.

Whitehead, S. D., & Ballard, D. H. (1989b). A role for anticipation in reactive systems that learn. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 354–357). San Mateo, CA: Morgan Kaufmann.

Whitehead, S. D., & Ballard, D. H. (1991). *A study of cooperative mechanisms for faster reinforcement learning* (Technical Report TR 365). Rochester, NY: University of Rochester, Department of Computer Science.

Williams, R. J. (1987). *Reinforcement-learning connectionist systems* (Technical Report NU-CCS-87-3). Boston, MA: Northeastern University, College of Computer Science.

Wilson, S. W. (1987). Classifier systems and the animate problem. *Machine Learning, 2*, 199–228.

Yee, R. C., Saxena, S., Utgoff, P. E., & Barto, A. G. (1990). Explaining temporal-differences to create useful concepts for evaluating states. *Proceedings of Ninth National Conference on Artificial Intelligence* (pp. 882–888). Cambridge, MA: MIT Press.