Advances in Reinforcement Learning and Their Implications for Intelligent Control

Steven D. Whitehead*, Richard S. Sutton† and Dana H. Ballard*

1 Introduction

What is an intelligent control system, and how will we know one when we see it? This question is hard to answer definitively, but intuitively what distinguishes intelligent control from more mundane control is the complexity of the task and the ability of the control system to deal with uncertain and changing conditions. A list of the definitive properties of an intelligent control system might include some of the following:

- **Effective:** The first requirement of any control system, intelligent or not, is that it generate control actions that adequately control the plant (environment). The adverb *adequately* is intentionally imprecise since effective control rarely means optimal, but more often means sufficient for the purposes of the task.
- **Reactive:** Equally important is that a system's control be timely. Effective control is useless if it is executed too late. In general, an intelligent control system cannot require arbitrarily long delays between control decisions. On the contrary, it must be capable of making decisions on a moments notice—upon demand. If extra time is available that can be exploited to improve the quality of a decision, so much the better, but control decisions must be available *any-time* [DB88].
- **Situated:** An intelligent control system should be tightly coupled to its environment, and its control decisions must be based on the immediate situation. A system must be capable of responding to unexpected contingencies and opportunities as they arise [AC87].
- **Adaptive** An intelligent control system must use its experience with the environment to improve its performance. Adaptability allows a system to refine an initially suboptimal control policy and to maintain effective control in the face of nonstationary environments.
- **Robust under Incomplete and Uncertain Domain Knowledge:** An intelligent control system must not depend upon a complete and accurate domain model. Even for relatively simple, narrow domains, it is extremely difficult to build models that are complete and accurate [Sha90]. If control depends upon a domain model, it must suffice that the model be incomplete and inaccurate. If domain knowledge is available, then the system should be capable of exploiting it, but it should not be a prerequisite for intelligent control. A system that uses a domain model that is learned incrementally through experience is to be preferred over one that relies upon a complete *a priori* model.
- **Perceptual Feasibility:** The information provided directly by a system's sensors is necessarily limited, and an intelligent control architecture must take this limitation into account. Instead of assuming that any and all information about the state of the environment is immediately available, intelligent system must be designed with limited but flexible sensory-motor systems. Control algorithms must consider actions for collecting necessary state information as well as actions for affecting change in the environment.
- **Mathematical Foundations:** To facilitate performance analysis, it is important that an intelligent control system be based on a framework that has a solid mathematical foundation.

While the above list is not particularly surprising or new, few if any control systems have been built that satisfy all of the requirements. For example, the problem-solving architectures that have been the dominate approach in Artificial Intelligence over the past twenty years fall well short of these goals. These problem-solving architectures are not particularly reactive, situated, adaptive, or robust in the face of incomplete and inaccurate domain models. They also tend to make unrealistic assumptions about the capabilities of the sensory system. While extensions and modifications to these architectures continue to be popular, other approaches are being considered as

*Department of Computer Science, University of Rochester, Rochester NY 14627
†GTE Laboratories Incorporated, Waltham, MA 02254

TH0333-5/90/0000/1289$01.00 © 1990 IEEE

well.

This paper focuses on control architectures that are based on reinforcement learning. In particular, we survey several recent advances in reinforcement learning that have substantially increased its viability as a general approach to intelligent control.

We begin by considering the relationship between reinforcement learning and dynamic programming, both viewed as methods for solving multi-stage decision problems [Wat89, BSW90]. It is shown that many reinforcement learning algorithms can be viewed as a kind of incremental dynamic programming; this provides a mathematical foundation for the study of reinfocrement learning systems. Connecting reinforcement learning to dynamic programming has also led to a strong optimal convergence theorem for one class of reinforcement learning algorithms and opens the door for similar analyses of other algorithms [Wat89].

Next we show how reinforcement learning methods can be used to go beyond simple trial-and-error learning. By augmenting them with a predictive domain model and using the model to perform a kind of incremental planning, their learning performance can be substantially improved. These control architectures learn both by performing experiments in the world and by searching a domain model. Because the domain model need not be complete or accurate, it can be learned incrementally through experience with the world [Sut90a, Sut90b, WB89, Whi89, Lin90].

Finally, we discuss active sensory-motor systems for feasible perception and how these systems interact with reinforcement learning. We find that, with some modification, many of the ideas from reinforcement learning can be successfully combined with active sensory-motor systems. The system then learns not only an overt control strategy, but also where to focus its attention in order to collect necessary sensory information [WB90b].

The results surveyed in this paper have been reported elsewhere (primarily in the machine learning literature). The objective here is to summarize them and to consider their implications for the design of intelligent control architectures.

2 Reinforcement Learning for Intelligent Control

What is reinforcement learning? A reinforcement learning system is any system that through interaction with its environment *improves its performance* by receiving *feedback* in the form of a scalar *reward (or penalty)* that is commensurate with the appropriateness of the response. By *improves its performance*, we mean that the system uses the feedback to adapt its behavior in an effort to maximize some measure of the reward it receives in the future. Intuitively, a reinforcement learning system can be viewed as a hedonistic automaton whose sole objective is to maximize the positive (reward) and minimize the negative (punishment).

Recent examples of controllers based on reinforcement learning include Barto *et al.*'s pole balancer [BSA83, Sut84], Grefenstette's simulated flight controller [Gre90], Lin's animats [Lin90], and Franklin's adaptive robot controllers [Fra88], among others.

2.1 Evaluating Reinforcement Learning

Reinforcement learning is emerging as an important alternative to classical problem-solving approaches to intelligent control because it possesses many of the properties for intelligent control that problem-solving approaches lack. In many respects the two approaches are complimentary and it is likely that eventual intelligent control architectures will incorporate aspects of both.[1]

Following is a discussion of the degree to which current reinforcement learning systems achieve each of the properties that we associate with intelligent control.

- **Effective:** Reinforcement learning systems are effective in the sense that they eventually learn effective control strategies. Although a system's initial performance may be poor, with enough interaction with the world it will eventually learn an effective strategy for obtaining reward. For the most part, the asymptotic effectiveness of reinforcement learning systems has been validated only empirically, however recent advances in the theory of reinforcement learning have yielded mathematical results that guarantee optimality in the limit for an important class of reinforcement learning systems [Wat89].
- **Reactive:** Decision-making in reinforcement learning systems is based on a policy function which maps situations (inputs) directly into actions (outputs) and which can be evaluated quickly. Consequently, reinforcement learning systems are extremely reactive.
- **Situated:** Reinforcement learning systems are situated because each action is chosen based on the current state of the world.
- **Adaptive:** Reinforcement learning systems are adaptive because they use feedback to improve their performance.
- **Incomplete and Uncertain Domain Knowledge:** Reinforcement learning systems do not depend upon internal domain models because they learn through trial-and-error experience with the world. However, when available, they can exploit domain knowledge by 1) using prior knowledge about the control task to determine a good

[1] To some extent this integration has already begun to occur, with the development of reinforcement learning system that learn and use internal domain models to improve overall performance.

initial policy [Fra88], 2) using an internal domain model to perform mental experiments instead of relying solely upon trial-and-error experiences, and 3) using a domain model to generalize the results of experiments in the world [YSUB90]. Also, because of the incremental nature of reinforcement learning, the models used need not be complete or accurate. This has lead to systems that profit from using models acquired through trial-and-error interaction with the world [Sut90a, Sut90b, Whi89, Lin90].

- **Perceptual Feasibility:** Most reinforcement learning systems have not addressed the issue of perceptual feasibility. However, recent results indicate that many of the ideas from reinforcement learning can be carried over (if indirectly) to systems that must actively control their sensory processes [WB90a, WB90b].
- **Mathematical Foundations:** Although reinforcement learning has been primarily an empirical science, there is a growing body of theory [BA85, Wil88, Sut88, Wat89] Advances relating reinforcement learning to dynamic programming are beginning to provide a solid mathematical foundation, as discussed below.

3 Reinforcement Learning as Incremental Dynamic Programming

Like much of artificial intelligence, reinforcement learning is primarily an empirical science, and the systems developed to solve reinforcement learning problems have been validated primarily through extensive simulation studies. However, this trend is changing, and a mathematical theory of reinforcement learning is beginning to emerge. The catalyst for this change has been the identification of the relationship between reinforcement learning and dynamic programming. In particular, Watkins has shown that certain reinforcement learning algorithms can be viewed as Monte Carlo versions of dynamic programming algorithms for solving multi-stage decision problems [Wat89].[2]

This connection has had three important implications for reinforcement learning:

1. It has lead to a strong optimality theorem concerning the asymptotic performance of an important class of reinforcement learning algorithms.
2. It has tied reinforcement learning to a well established mathematical foundation on which further analytical studies can be based.
3. It has clarified our intuitive understanding of reinforcement learning and has directly contributed to the development of extended architectures that are more general and outperform previous architectures (cf. Sections 4 and 5).

[2] To our knowledge, Werbos first made the connection between reinforcement learning and the theory of multi-stage decision problems. However, Watkins solidified the connection and was first to obtain theoretical results.

3.1 Multi-stage Decision Problems

Multi-stage decision problems are modeled as Markov decision processes. A Markov decision process is defined by the tuple $(\mathbf{S}, \mathbf{A}, T, R)$, where $\mathbf{S}$ is the set of possible states the world can occupy; $\mathbf{A}$ is the set of possible actions a controller may execute to change the state of the world; T is the state transition function; and R is the reward function. Usually, $\mathbf{S}$, and $\mathbf{A}$ are discrete and finite. In a discrete-time Markov decision process, time advances by discrete, unit length quanta; $t = 0, 1, 2, \ldots$. At each time step the world occupies one state, $s_t \in \mathbf{S}$ and a time step occurs when the controller applies an action, $a_t \in \mathbf{A}$. The result of executing an action is a new state s_{t+1} and the receipt of a reward r_{t+1}. The state transition function, T, models the effects of applying different actions in different states and maps state-action pairs into a resulting new state. In general, transitions are probabilistic so that applying action a in state s yields a new state $s' = T(s, a)$ that is drawn from a probability distribution over $\mathbf{S}$. The probabilities that govern the transition function depend only upon the action selected and the state in which it was applied. These probabilities are assumed to be known and are denoted by $P_{x,y}(a)$ where

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

As with transitions, rewards are generated probabilistically and R is a probabilistic function of the state-action pair executed. The distributions governing R depend only upon the state-action pair executed, and are assumed to be known.

Given a description of a Markov decision process, the objective is to find a control policy (i.e. a mapping from states to actions) that, when executed by the controller, maximizes some measure of the cumulative reward received over time.

There are numerous measures of cumulative reward. One of the most common is a measure based on a discounted sum of the reward received over time. This sum is called the *return* and for time t is defined as

$$\mathbf{r}_t = \sum_{n=0}^{\infty} \gamma^n r_{t+n+1} \tag{2}$$

where γ is a discount factor between 0 and 1. Because the process is stochastic, the objective is to find a decision policy that maximizes the *expected return*.

For a fixed policy π, define $V_\pi(x)$ to be the expected return given that the process begins in state x and follows policy π thereafter. V_π is called the *utility function* for policy π and can be used to define optimality criteria. The objective is to find a policy whose utility function is uniformly maximal for every state. That is, to find an optimal policy π^* such that

$$V_{\pi^*}(x) = \max_{\pi}(V_\pi(x)) \qquad \forall_{x \in \mathbf{S}} \tag{3}$$

The *optimality theorem* from the theory of multi-stage decision problems guarantees that for a stationary, discrete time, discrete state Markov decision process there exists a deterministic decision policy that is optimal. Furthermore, a policy π is optimal if and only if:

$$Q_\pi(x, \pi(x)) = \max_{b \in \mathbf{A}}(Q_\pi(x, b)) \qquad \forall_{x \in \mathbf{S}} \qquad (4)$$

where $Q_\pi(x, a)$, called the *action-value* for the state-action pair (x, a), is defined as the return the system expects to receive given that it starts in state x, applies action a next, and then follows policy π thereafter [Bel57, Ber87].

3.2 Policy Optimization by Dynamic Programming

Two of the most important dynamic programming methods for computing the optimal policy for a given Markov decision process are *policy iteration* and *value iteration*. Policy iteration begins with an arbitrary policy and monotonically improves it until it converges on an optimal policy. The principle idea is simply to choose a policy, π; compute V_π, the expected return associated with that policy; and then improve the policy by replacing actions whose local counterparts outperform them. In value iteration, the optimal utility function is directly computed without going through a series of suboptimal policies. Once the optimal utility function has been obtained it is then straightforward to compute the optimal policy via Equation 4.

3.3 The Relationship Between Reinforcement Learning and Multi-Stage Decision Problems

There is a close relationship between reinforcement learning and using dynamic programming to solve multi-stage decision problems. In both the world is characterized by a set of states, a set of possible actions, and a reward function. In both the objective is to find a decision policy that maximizes the cumulative reward received over time. There is an important difference though. When solving a multi-stage decision problem, the analyst (presumably the designer of the eventual control system) has a complete (albeit stochastic) model of the environment's behavior. Given this information, the analyst can compute the optimal control policy with respect to the model, as outlined above. In reinforcement learning, the set of states, and the set of possible actions is known *a priori*, but the effects of action on the environment and on the production of reward is not. Thus, the designer cannot compute an optimal policy *a priori*. Instead the control system must learn an optimal policy by experimenting in the environment.

3.4 Q-learning

In addition to recognizing the intrinsic relationship between reinforcement learning and dynamic programming, Watkins has made an important contribution to reinforcement learning by suggesting a new learning algorithm called *Q-learning*. The significance of Q-learning is that one version of it, *1-step Q-learning*, when applied to a Markov decision process, can be shown to converge to the optimal policy, under appropriate conditions. 1-step Q-learning is the first reinforcement learning algorithm to be shown convergent to the optimal policy for decision problems involving delayed reward.

The connections between Q-learning and dynamic programming are strong: 1-step Q-learning is motivated directly by value-iteration and its convergence proof is based on a generalization of the convergence proof for value-iteration [Wat89].

In value-iteration, the optimal policy is obtained in the limit by solving a series of finite horizon tasks. The equations for computing each cycle in the iteration, (i.e. V^n from V^{n-1}) are [Wat89]:

$$Q^n(x, a) = \mathbf{E}[R(x, a)] + \gamma \mathbf{E}[V^{n-1}(T(x, a))] \qquad (5)$$

$$V^n(x) = \max_{a \in \mathbf{A}} Q^n(x, a) \qquad (6)$$

$$\pi^n(x) = a \text{ such that } Q^n(x, a) = V^n(x) \qquad (7)$$

where π^n, V^n, and Q^n are the optimal policy, value, and action-value functions for the n-stage task, respectively.

In reinforcement learning these equations cannot be solved iteratively since the statistics required to compute the expectations are unavailable. That is, $\mathbf{E}[R(x, a)]$ and $P_{x,y}(a)$ (needed to compute $\mathbf{E}[V^{n-1}(T(x, a))]$) are unknown. The principle of 1-step Q-learning is to solve these equations incrementally by using experience gained through interactions with the actual environment to estimate the expectations in Equation 5. Replacing the expectations in Equation 5 with the values observed during a trial, leads to the following updating rules:

$$\tilde{Q}_{t+1}(x_t, a_t) = (1-\alpha)\tilde{Q}_t(x_t, a_t) + \alpha[r_t + \gamma \tilde{V}_t(x_{t+1})] \qquad (8)$$

$$\tilde{V}_{t+1}(x) = \max_{a \in \mathbf{A}} \tilde{Q}_{t+1}(x, a) \qquad (9)$$

$$\tilde{\pi}_{t+1} = a \text{ such that } \tilde{Q}_{t+1}(x, a) = \tilde{V}_{t+1}(x) \qquad (10)$$

where $\tilde{V}_t$ and $\tilde{Q}_t$ are the system's estimates for the optimal utility and action-value functions at time t. Notice that the bracketed term on the right hand side of Equation 8 is an estimate of the system's future return that is based on the actual results of executing 1-step, a_t in state x_t at time t (i.e., it estimates the term on the right-hand side of Equation 5). For n-step Q-learning, this 1-step estimate is replaced by its n-step counterpart, based on the actual results of executing n actions. In this case, the bracketed term becomes:

$$[r_t + \gamma r_{t+1} + \ldots \gamma^{n-1} r_{t+n-1} + \gamma^n \tilde{V}_t(x_{t+n})] \qquad (11)$$

Watkins has shown that any 1-step Q-learning system that 1) decreases its learning rate at an appropriate rate and 2) tests each state-action pair infinitely often over the coarse of its lifetime is guaranteed to converge to an optimal policy [Wat89].

4 Beyond Trial and Error Learning: Incremental Planning

Reinforcement learning and dynamic programming lie at opposite ends of a spectrum. At one end, reinforcement learning systems *learn* control policies based solely on trial-and-error interactions with the environment. No dynamical model of the world is needed or used. At the other end, dynamic programming is used to *compute* control policies based solely on a complete dynamical model of the Markov decision process. Dynamic programming does not rely on actual experience with the world.

The principle advantage of dynamic programming is that, if a problem can be specified in terms of a Markov decision process, then it can be analyzed and an optimal policy obtained *a priori*. Other than computational complexity, the two principle *dis*advantages of dynamic programming are 1) for many tasks, it is difficult to specify the dynamical model and 2) because dynamic programming determines a fixed control policy *a priori*, it does not provide a mechanism for adapting the policy to compensate for non-stationary dynamics or modeling errors.

Reinforcement learning has the complementary adavantages: 1) it does not require a prior dynamical model of any kind, but learns based on experience gained directly from the world, and 2) to some degree, it can track the dynamics of non-stationary systems. The principle disadvantage of reinforcement learning is that in general many trials (repeated experiences) are required to learn an optimal control strategy, especially if the system starts with a poor initial policy.

This suggest that the respective weaknesses of these two approaches may be overcome by integrating them. That is, if a complete, possibly inaccurate, model of the task is available *a priori*, dynamic programming can be used to develop the initial policy for a reinforcement learning system. A reasonable initial policy can substantially improve the system's initial performance and reduce the time required to reach an acceptable level of performance [Fra88]. Conversely, adding an adaptive, reinforcement learning component to an otherwise fixed controller whose policy is determined by dynamic programming can compensate for an inaccurate model.

Although this simple approach helps to mitigate the poor initial performance of *naive* reinforcement learning systems, it does not in itself improve their overall learning *rate*. It has been proposed by Sutton and Whitehead that learning rate can be improved and other advantages obtained if reinforcement learning systems are, in addition, augmented with a predictive model of the environment [Sut90a, Sut90b, WB89, Whi89]. These *Dyna* architectures are based on the idea that planning is like trial-and-error learning from hypothetical experience. That is, the model is used to construct hypothetical experiences, and then these are learned from just as if they had actually happened.

An outline of a Dyna architecture is shown in Figure 1. The two main components are a reinforcement learning subsystem, which can employ any number of reinforcement learning algorithms (e.g Q-learning, AHC algorithms, Bucket Brigade, etc), and a predictive model, which mimics the one-step input-output behavior of the world. The significance of the internal model in Dyna architectures is that it provides a fast and inexpensive mechanism for propagating the effects of actual experience throughout the system.

To see how an internal model can help, consider a 1-step Q-learning system that at time t, applies action a_t in state x_t and as a result obtains the new state x_{t+1} and receives the reward r_{t+1}. In 1-step Q-learning, the only immediate effect of this experience is to change the functional values associated with state x_t. That is, $Q(x_t, a_t)$, $V(x_t)$, and $\pi(x_t)$ stand to change, but no other values will. Eventually the ramifications of the experience will be propagated to states other than x_t. For example, changes in $V(x_t)$ effect the value estimates of other neighboring states that are causally connected to x_t (i.e., can immediately proceed x_t). These states, in turn, will effect the utility estimates of their causally connected neighbors, and so on. In general, a single experience can have ramifications that effect the policy and utility value of every state. What makes 1-step Q-learning slow is that propagation occurs only one step at a time. The transitions in a sequence must be traversed $O(n)$ times, in order for the effects of an event to be propagated back n-stages.[3]

Using an internal model to perform hypothetical experiments is a fast and inexpensive mechanism for propagating utility information throughout the system. Hypothetical reasoning is fast, because the effects of actions can be simulated faster than they can be performed in the real world, and because hypothetical experiments need not be tied to the current state, but can be performed in any state that can be imagined. Hypothetical reasoning is also inexpensive because the reward and punishment received by the system is imaginary. As a result, the amount of time *actually* exposed to dangerous situations and the cumulative punishment received for performing badly is reduced.

In addition to improving learning rate, Dyna architectures exhibit a number of other important properties. First hypothetical experiences in Dyna archi-

[3] Other reinforcement learning algorithms exist that use memory (or "eligibility") traces to propagate back the effects of an action to the states that preceed it. Most notable are the algorithms based on Temporal Difference Methods [Sut88]. These algorithms mitigate the propagation problem to some degree. However, they do not do a complete job because they only effect the sequence of states that immediately proceeded the event. Other trial-and-error experiments are necessary to propagate the changes to other causally related states not found in the experienced sequence.

tectures are incremental. A hypothetical experience can be as short as the simulation of one action and yet be completely effective. Thus, Dyna architectures provide a means for integrating reactive control with search based planning. Also Dyna architectures do not depend upon complete and accurate domain models. Instead they can use partial models that are learned as experience is gained in the world.

5 Beyond Perfect Perception: Active Representations

For the most part, reinforcement learning research has avoided issues of perception. In most reinforcement learning systems the sensory system is either trivially simple or abstracted out of the model altogether. The usual assumption, as in a Markov decision process, is that after each action the system observes the state of the world. The "state" of the world is defined by the values of the system's sensory inputs, and usually these inputs are carefully chosen by the system designer.

In more realistic learning tasks, complete knowledge of the task cannot be exploited in the design of the sensory system. In this case, the system must learn which aspects of the world are relevant to the task on its own. Before learning to solve the problem, the system must learn to represent it. One approach to this problem is to build a sensory system that is as complete as possible. A more practical approach is to consider systems that can flexibly sense different aspects of the world, but that on a moment-by-moment basis only register a limited amount of information. For example, one can imagine an autonomous robot that possesses a large repertoire of sensory routines that it can use to analyze the world (say 100 or so), but because of time, space, and processing constraints, it can only afford to apply a few at a time (say less than 10) [Ull84, TS90]. Deictic (or functional indexical) representations offer another example of this *active* approach to perception [AC87, Agr88, Cha90]. In a deitic representation, the system is capable of attending to only a limited number of objects at a time, and the properties of those objects form the basis of the system's sensory inputs. By changing its focus of attention, the system can represent different parts of the world and obtain a variety of representations for the same situation.

Whitehead and Ballard have investigated control architectures that integrate active perception and reinforcement learning [WB90b]. In the following subsections we present their three main contributions: 1) an abstract model that formalizes the functional relationships that exist between the world, the sensory-motor system, and the embedded decision system; 2) analyses and demonstrations that the integration of active perception and reinforcement learning is non-trivial, due to aliasing in the representation of world states; and 3) a new reinforcement learning algorithm that overcomes the difficulties caused by active perception for a restricted class of tasks.

5.1 The Formal Model

The formal model describing a learning agent and the world in which it is embedded is shown in Figure 2. The external world is modeled as a Markov decision process (whose statistical parameters are unknown to the agent). The decision process is characterized by the tuple $(\mathbf{S}_E, \mathbf{A}_E, T, R)$ where $\mathbf{S}_E$ is the set of external world states, $\mathbf{A}_E$ is the set of external (overt) actions the agent can perform on the world, T is the transition function, R is the reward function.

The agent has two major subsystems: an active sensory-motor subsystem and a decision subsystem. The sensory-motor system implements three functions: 1) a perceptual function $\mathcal{P}$; 2) an internal configuration function $\mathcal{I}$; and 3) a motor function $\mathcal{M}$. The purpose of the sensory-motor subsystem is to ground internal perceptions and actions in the external world. On the sensory side, the system translates external world states 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, the relationship between external world states and the agent's internal representation is modeled by the *perceptual function* $\mathcal{P}$, which maps the set of possible world states $\mathbf{S}_E$ and the set of possible sensory-motor configurations $\mathbf{C}$ onto the set of possible internal representations $\mathbf{S}_I$. On the motor side, the agent has a set of *internal motor commands*, $\mathbf{A}_I$, that affect the model in two ways: they can either change the state of the external world (by being translated into external actions, $\mathbf{A}_E$), 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 commands that change the configuration of the sensory-motor system are called perceptual actions. As with perception, the configuration of the sensory-motor system relativizes the effects of internal commands. This dependence is modeled by the functions $\mathcal{M}$ and $\mathcal{I}$, which map internal commands and sensory-motor configurations into actions in the external world and into new sensory-motor configurations, respectively.

The remaining component of the agent is the decision subsystem. This subsystem is like a homunculus that sits inside the agent's head and controls its actions. The decision subsystem corresponds to the reinforcement learning systems discussed previously except now it is *embedded* inside the agent and buffered from the external world by the sensory-motor system. 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 $\mathcal{B}$ that maps sequences of internal states and rewards $(\mathbf{S}_I \times \Re)^*$ into internal actions, $\mathbf{A}_I$.

5.2 Perceptual Aliasing

Interposing an active sensory-motor system between the world and the decision system can lead to decision problems which cannot be learned using standard reinforcement learning techniques. The problem arises from the many-to-many mapping between states in the external world and states in the internal representation. That is, a state $s_e \in S_E$ in the world may map to several internal states, depending upon the configuration of the sensory-motor system. More importantly, a single internal state, $s_i \in S_I$, may represent multiple world states. This overlapping between the world and the agent's internal representation is called *perceptual aliasing* [WB90b].

Perceptual aliasing can transform a problem that is Markovian into one that is not. Intuitively, perceptual aliasing interferes with reinforcement learning by allowing the decision subsystem to confound (perceive as the same) external world states that may have different utility values. For example, suppose that depending upon the configuration of the sensory-motor system, an internal state s_a can represent one of two world states, S_{E1} or S_{E2}. In reinforcement learning, utility values are estimated by averaging the rewards accumulated over time. Assuming the system performs optimally, the utility estimate learned for the internal state s_a (denoted $\tilde{V}_I(s_a)$) will correspond to a sampled average of the utilities for S_{E1} and S_{E2}. If both states are encountered equally often then the utility estimate for state s_a will be approximately equal to their arithmetic mean

$$\tilde{V}_I(s_a) \approx \frac{V_{\pi_E^*}(S_{E1}) + V_{\pi_E^*}(S_{E2})}{2} \qquad (12)$$

If the utility values for S_{E1} and S_{E2} are the same, then the learned estimate for s_a will reflect the actual return the system can expect to receive whenever it finds itself in state s_a. However, when the utility values for S_{E1} and S_{E2} differ (say $V_{\pi_E^*}(S_{E1}) \ll V_{\pi_E^*}(S_{E2})$) then $\tilde{V}(s_a)$ will fail to accurately estimate the expected utility associated with the current world state. If the external world is in state S_{E1} then $\tilde{V}(s_a)$ will overestimate the expected return; if the world is in state S_{E2} then $\tilde{V}_I(s_a)$ will underestimate the expected return. This potential for a mismatch in estimating the utility of world states, caused by perceptual aliasing, interferes with the decision system's ability to learn the optimal policy.

5.3 The Lion Algorithm

Whitehead and Ballard have recently proposed a new learning algorithm, called the *lion algorithm* that overcomes the difficulties caused by perceptual aliasing for deterministic tasks. The lion algorithm is based on the notion of *consistent internal states*. Intuitively, an internal state is *consistent* if all the external world states it represents are the same in the following sense: 1) they all have the same optimal utility values, and 2) the optimal actions for each map into the same internal command to be executed by the decision subsystem. In the above example, s_a is inconsistent because $V_{\pi_E^*}(S_{E1}) \neq V_{\pi_E^*}(S_{E2})$.

The key property of consistent internal states in deterministic tasks is that whenever one is encountered, the optimal return from that point forward is fixed and independent of the actual state of the external world. For inconsistent internal states the optimal return depends upon the actual state of the world and cannot be absolutely determined from knowledge of the inconsistent internal state. In general, each internal state defines an equivalence class of external world states. A consistent internal state is useful because (by definition) it guarantees that every world state in its equivalence class has the same utility and same optimal action command. Inconsistent internal states are not useful for predicting the utility of the current situation or the optimal action to be executed next.

The principle idea of the lion algorithm is to actively configure the sensory-motor system so that at each time step the system's internal representation is consistent and the overt action the system chooses to execute next is based upon the estimates of consistent representations. Since perceptual aliasing interferes with reinforcement learning by injecting inconsistent states into the decision process, the lion algorithm actively detects and prevents them from participating in the decision making process. The basic steps in the control cycle are as follows:

1. *Execute a series of perceptual actions in order to collect a set of internal representations for the current world state.*
2. *Choose one internal state that is believed to be consistent from the set collected in step 1. Call this state the lion.*
3. *Choose the next overt action to execute based on the policy value for the lion.*
4. *Use the lion state, along with the return received on the last time step to update the utility and action-value estimates for the previous cycle.*

The key operation in the lion algorithm is to select, for each situation, one internal state that is believed to be consistent (i.e., Step 2 above). This operation is achieved by differentiating between consistent and inconsistent states based on the observation that, for deterministic tasks, the variance in the utility estimates for inconsistent states will always be non-zero, whereas the variance for consistent states will tend to zero over time. For details see [WB90b].

The lion algorithm has been demonstrated in a system that learns to solve a simple class of block manipulation tasks. The significance of the demonstration is not the difficulty of the task *per se*, but that the system employed an active (deictic) sensory-motor system, and learned not only how to solve the problem but also how to control its sensory apparatus to

attend to the objects relevant to the task. Although the results are preliminary, the idea of learning task-dependent representations using reinforcement learning, active sensory-motor systems, and the notion of consistency is an important step towards the development of ever more autonomous learning agents.

6 Summary

In this paper we have surveyed a number of recent advances that have contributed to the viability of reinforcement learning approaches to intelligent control. These advances include the formalization of the relationship between reinforcement learning and dynamic programming, the use of internal predictive models to improve learning rate, and the integration of reinforcement learning with active perception. Based on these and other results, we conclude that control architectures based on reinforcement learning are now in a position to satisfy many of the criteria associated with intelligent control.

References

[AC87] Philip E. Agre and David Chapman. Pengi: an implementation of a theory of activity. In *AAAI*, pages 268–272, 1987.

[Agr88] Philip E. Agre. *The Dynamic Structure of Everyday Life.* PhD thesis, MIT Artificial Intelligence Lab., 1988. (Tech Report No. 1085).

[BA85] A. G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–375, 1985.

[Bel57] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[Ber87] D. P. Bertsekas. *Dyanmic Programming: Deterministic and Stochastic Models.* Prentice-Hall, 1987.

[BSA83] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuron-like elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[BSW90] Andrew B. Barto, Richard S. Sutton, and Chris J. C. Watkins. Learning and sequential decision making. In M. Gabrial and J. W. Moore, editors, *Learning and Computational Neuroscience.* MIT Press, Cambridge, MA, 1990. (Also COINS Tech Report 89-95, Dept. of Computer and Information Sciences, University of Massachusetts, Amherst, MA 01003).

[Cha90] David Chapman. *Vision, Instruction, and action.* PhD thesis, MIT Artificial Intelligence Laboratory, 1990. (Technical Report 1204).

[DB88] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings AAAI-88*, pages 49–54, 1988.

[Fra88] Judy A. Franklin. Refinement of robot motor skills through reinforcement learning. In *Proceedings of the 27th IEEE Conference on Decision and Control*, Austin, TX, December 1988.

[Gre89] John J. Grefenstette. Incremental learning of control strategies with genetic algorithms. In *Proceedings of the Sixth International Workshop on Machine Learning.* Morgan Kaufmann, June 1989.

[Lin90] Long-Ji Lin. Self-improving reactive agents: Case studies of reinforcement learning frameworks. In *Proceedings of the First International Conference on the Simulation of Adaptive Behavior*, September 1990.

[Sha90] Steve Shafer. Why we can't model the physical world. (for the 25th anniversary of the CMU CS Dept.), September 1990.

[Sut84] Richard S. Sutton. *Temporal Credit Assignment In Reinforcement Learning.* PhD thesis, University of Massachusetts at Amherst, 1984. (Also COINS Tech Report 84-02).

[Sut88] Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[Sut90a] Richard S. Sutton. First results with DYNA, an integrated architecture for learning, planning, and reacting. In *Proceedings of the AAAI Spring Symposium on Planning in Uncertain, Unpredictable, or Changing Environments*, 1990.

[Sut90b] Richard S. Sutton. Integrating architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, TX, 1990. Morgan Kaufmann.

[TS90] Ming Tan and Jeffery C. Schlimmer. Two case studies in cost-sensitive concept acquistition. In *Proceedings of AAAI-90*, 1990.

[Ull84] Shimon Ullman. Visual routines. *Cognition*, 18:97–159, 1984. (Also in: Visual Cognition, S. Pinker ed., 1985).

[Wat89] Chris Watkins. *Learning from delayed rewards.* PhD thesis, Cambridge University, 1989.

[WB89] Steven D. Whitehead and Dana H. Ballard. A role for anticipation in reactive systems that learn. In *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, NY, 1989. Morgan Kaufmann.

[WB90a] Steven D. Whitehead and Dana H. Ballard. Active perception and reinforcement learning. In *Proceedings of the Seventh International Conference on Machine Learning*. Morgan Kaufmann, June 1990. (Also to appear in Neural Computation).

[WB90b] Steven D. Whitehead and Dana H. Ballard. Learning to perceive and act. Technical Report TR 331 (revised), Computer Science Dept., University of Rochester, 1990. (Also submitted to Machine Learning).

[Whi89] Steven D. Whitehead. Scaling in reinforcement learning. Technical Report TR 304, Computer Science Dept., University of Rochester, 1989.

[Wil88] R. J. Williams. Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA, 1988.

[YSUB90] Richard C. Yee, Sharad Saxena, Paul E. Utgoff, and Andrew G. Barto. Explaining temporal-differences to create useful concepts for evaluating states. In *Proceedings of AAAI-90*, 1990.

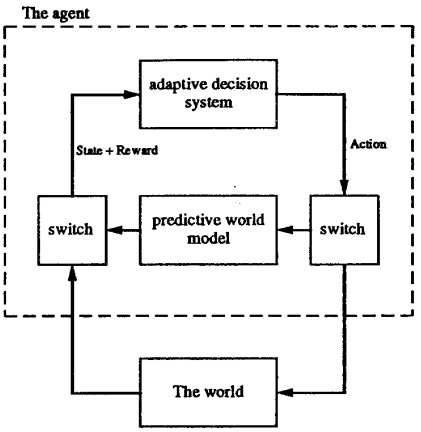

Figure 1: Dyna architectures are organized around two basic components: an adaptive decision system (used for control) and an adaptive internal model (used to predict the input-output behavior of the world). The model is used to perform hypothetical experiments. A switch is used to modulate the decision system's interaction with the world and the model.

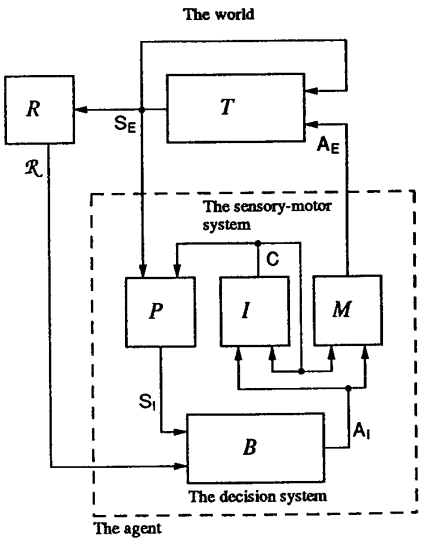

Figure 2: A formal model for an agent with an embedded learning subsystem and an active sensory-motor subsystem.