Real-time Learning and Control using Asynchronous Dynamic Programming

Andrew G. Barto, Steven J. Bradtke & Satinder P. Singh
Computer Science Department
University of Massachusetts

Technical Report 91-57

August 1991

ACKNOWLEDGEMENTS:

The authors thank Richard Yee, Vijay Gullapalli, Brian Pinette, and Jonathan Bachrach for helping to clarify the relationships between heuristic search and control. We thank Rich Sutton, Chris Watkins, Paul Werbos, and Ron Williams for sharing their fundamental insights into this subject through numerous discussions, and we further thank Rich Sutton for first making us aware of Korf's work and for his very thoughtful comments on the manuscript. This research was supported by grants to A.G. Barto from the National Science Foundation (ECS-8912623) and the Air Force Office of Scientific Research, Bolling AFB (AFOSR-89-0526).

Abstract

Learning methods based on dynamic programming (DP) are receiving increasing attention in artificial intelligence. Researchers have argued that DP provides the appropriate basis for compiling planning results into reactive strategies for real-time control, as well as for learning such strategies when the system being controlled is incompletely known. We extend the existing theory of DP-based learning algorithms by bringing to bear on their analysis a collection of relevant mathematical results from the theory of asynchronous DP. We present convergence results for a class of DP-based algorithms for real-time learning and control which generalizes Korf's Learning-Real-Time-A* (LRTA*) algorithm to problems involving uncertainty. We also discuss Watkins' *Q-Learning* algorithm in light of asynchronous DP, as well as some of the methods included in Sutton's *Dyna* architecture. We provide an account that is more complete than currently available of what is formally known, and what is not formally known, about the behavior of DP-based learning algorithms. A secondary aim is to provide a bridge between AI research on real-time planning and learning and relevant concepts and algorithms from control theory.

1 Introduction

The increasing interest of artificial intelligence (AI) researchers in systems embedded in environments demanding real-time performance is narrowing the gulf between problem solving and control engineering. Similarly, machine learning techniques suited to embedded systems are becoming more comparable to methods for adaptive control of dynamical systems. Although there continues to be substantial interest in bringing AI methods to bear on control problems, often under the banner of "intelligent control" (e.g., ref. [55]), in this article our interest is less in hybrids of knowledge-based systems and conventional controllers than in the fundamental integration of certain AI and control engineering methods at the algorithmic level. Whereas AI has focused on problems having relatively little mathematical structure, control theorists have studied more restrictive classes of problems but have developed correspondingly more detailed theories. The algorithms described in this article exploit the complementary properties of algorithms developed in these disciplines.

We describe the relationship between some recent developments in heuristic search and optimal and adaptive control, and we examine the implications of this relationship, especially with respect to learning. We adopt a framework based on the theory of dynamic programming (DP) as developed for applications in control engineering. Learning and planning methods based on DP are receiving increasing attention in AI as researchers argue that DP provides the appropriate basis for compiling planning results into reactive strategies that can be used for real-time control, as well as for learning such strategies when the system being controlled is incompletely known. Watkins [67] and Werbos [72, 73] proposed using incremental versions of DP algorithms for this purpose, and Sutton's *Dyna* architecture for learning, planning, and reacting [58, 59] is based on these principles. Although rigorous results have been developed for some of these methods, most of this research concerns methods whose theoretical basis has not yet been fully developed.

In this article we extend the existing theory of DP-based algorithms for real-time learning and control by bringing to bear on their analysis a collection of relevant mathematical results. This permits us to give a more complete account of what is known formally about the behavior of these algorithms, as well as a clear account of what is not known. A secondary aim of this article is to provide a bridge between AI research on real-time planning and learning and relevant concepts and algorithms from control theory. Specifically, we use the theory of asynchronous DP developed by Bertsekas [7] and Bertsekas and Tsitsiklis [9] to obtain convergence results for a class of algorithms for real-time learning and control. This class of algorithms includes Korf's [32] Learning-Real-Time-A* (LRTA*) algorithm and a generalization of it applicable to problems involving uncertainty. Some of our theory follows from combining Korf's results with results from the theory of asynchronous DP. We also discuss methods applicable when information is lacking not only about a problem's solution, but also about the structure of the problem itself. To use the control theorists' terminology, these are *adaptive control problems*. Although we prove no new convergence results for algorithms applicable to this adaptive case, we precisely discuss the issues such results must address. We discuss Watkins' influential *Q-Learning* algorithm [67] in light of asynchronous DP, as well as some of the methods included in Sutton's *Dyna* architecture [58].

Because the reader is unlikely to be familiar with all of the contributing lines of research, we provide the necessary background in Section 2, followed in Section 3 by a discussion of the proper relationship between some concepts from AI and control theory. Development of the theoretical material occupies Sections 4 through 9, with an introduction to conventional DP algorithms occupying Section 5. There are two major parts to this theoretical development. The first part (Sections 5 and 6) concerns problems in which there exists a complete and accurate model of the problem being solved. The second part (Section 7) concerns the additional complexity present when such a model is lacking, i.e., in the adaptive case. Despite the generality of this theory, other learning algorithms based on DP have been studied to which it does not apply, as discussed in Section 8. In Section 9 we discuss some of the issues that practical implementations of these algorithms must address. In Section 10 we present an example problem that we use to illustrate the algorithms. Simulation results are presented in Section 11. We conclude in Section 12 with an appraisal of the significance of our approach to problems of learning and control and discuss some of the open problems.

2 Background

The method Samuel [47, 48] used to learn a heuristic evaluation function for the game of checkers has been a major influence on research leading to the algorithms on which this article focuses. His method updated board evaluations by comparing an evaluation of the current board position with an evaluation of a board position likely to arise later in the game:

> ... we are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board position of the chain of moves which most probably occur during actual play. (Samuel [47].)

As a result of this process of "backing up" board evaluations, the evaluation function should improve in its ability to evaluate the long-term consequences of moves. In one version of this algorithm, Samuel represented the evaluation function as a weighted sum of numerical features and adjusted the weights based on an error derived from comparing evaluations of current and predicted board positions.

Because of its compatibility with connectionist learning algorithms, this approach was refined and extended by Sutton [56, 57] and used heuristically in a number of single-agent problem-solving tasks (e.g., Barto, Sutton, and Anderson [2], Anderson [1], and Sutton

[56]). The algorithm was implemented as a neuron-like connectionist element called the Adaptive Critic Element [2]. Sutton [57] called these algorithms *Temporal Difference* (TD) methods and obtained some formal results about their convergence. Following the proposals of Klopf [30, 31], Sutton and Barto [60, 61, 62] developed these methods as models of animal learning. Minsky [42, 43] discussed similar ideas in the context of the credit assignment problem for reinforcement learning systems, and Hampson [21] independently developed some of these ideas and related them to animal behavior. Holland's [22] bucket-brigade algorithm for assigning credit in his classifier systems is also closely related to Samuel's method. Researchers in machine learning and connectionism use the term *reinforcement learning* to describe this general class of learning problems and algorithms. Sutton, Barto, and Williams [63] discuss reinforcement learning from the perspective of DP and adaptive control.

Another line of research leading to algorithms for backing up evaluations comes from the theory of optimal control, where DP provides important solution methods. As applied to control problems, DP (introduced in 1957 by Bellman [6]) consists of methods for successively approximating optimal evaluation functions for both deterministic and stochastic optimal control problems. In its most general form, DP applies to optimization problems in which the costs of objects in the search space have a recursive structure that can be exploited to find a minimum-cost object without performing exhaustive search. Kumar and Kanal [34] discuss DP at this level of generality and relate it to a variety of search algorithms used in AI. However, we restrict attention to DP as it applies to problems in which the objects are state sequences that can be generated in problem-solving or control tasks. DP algorithms solve these optimization problems by solving recurrence relations instead of conducting a search in the space of state sequences. Backing up state evaluations is the basic step of iterative procedures for solving these recurrence relations.

Although DP algorithms avoid exhaustive search in the state-sequence space, they are still exhaustive by AI standards because they require the repeated generation and expansion of all possible states; that is, they require many explicit computational steps for every state and its possible successor states. For this reason DP algorithms have not played a significant role in AI. Indeed, an important feature distinguishing heuristic search and DP algorithms is that heuristic search algorithms abandon the objective of optimality in exchange for relative computational efficiency in problems with extremely large numbers of states. But DP algorithms are relevant to learning in a way that heuristic search algorithms are not because they work by updating *stored* evaluations of the states; in effect, they cache the results of repeated shallow searches in a permanent data structure.

Despite their exhaustive nature, it is possible to rearrange the computational steps of DP algorithms so that they can be applied *during* control or real-time problem solving. This is the basis of the learning algorithms we describe in this article. In most cases, convergence to an optimal evaluation function still requires multiple generation and expansion of all possible states, but performance improves incrementally while this is being accomplished.

This perspective was taken by Werbos [70], who proposed a method similar to that used in the Adaptive Critic Element within the framework of DP. He called this approach *Heuristic Dynamic Programming* and has written extensively about it (e.g., refs. [71, 72, 73, 75]). Related algorithms have been discussed by Witten [80, 81], and more recently, Watkins [67] extended Sutton's TD algorithms and developed others by explicitly utilizing the theory of DP. He used the term *Incremental Dynamic Programming* to refer to this class of algorithms and discussed many examples. Williams and Baird [79] theoretically analyzed a variety of additional DP-based algorithms suitable for real-time application.

Sutton [58] proposed the *Dyna* architecture for integrating learning and planning by means of incremental DP algorithms. The key idea in *Dyna* is that one can perform the computational steps of an incremental DP algorithm sometimes using information obtained from state transitions actually taken by the system being controlled, and sometimes from hypothetical state transitions simulated using a model of this system. This approach interleaves phases of planning—performed using hypothetical state transitions—with the generation of control actions in order to satisfy time constraints. The underlying DP algorithm provides the means for compiling the resulting information into a form that can be used efficiently for directing the course of action. Another aspect of *Dyna* is that the system model can be refined through a learning process deriving training information from the state transitions actually observed during control. Even without this on-line model refinement, however, interleaving the steps of a DP algorithm with the generation of control actions has implications for planning in AI, as discussed by Sutton in ref. [59].

Many researchers have made additional contributions to using incremental DP algorithms for planning and control (e.g., Barto and Singh [5], Chapman and Kaelbling [12], Dayan [15, 14], Jalali and Ferguson [24], Kaelbling [26], Lin [39, 38], Moore [44, 45], Schmidhuber [49, 50], Singh [53], Singh [54], Tan [64], Thrun and Möller [65], Utgoff and Clouse [66], Whitehead [77], Whitehead and Ballard [78], Wixson [82]). Although this general approach can be applied to problems involving continuous time and/or state spaces, we restrict attention to discrete-time problems with finite sets of states and control actions because of their relative simplicity and closer relationship to the non-numeric problems usually studied in AI. This excludes various "differential" approaches, which make use of gradient descent and algorithms related to the connectionist error-backpropagation algorithm (e.g., Jacobson and Mayne [23], Jordan and Jacobs [25], Werbos [69, 74]). We do, however, include stochastic problems, i.e., problems involving probabilistic models of uncertainty. The stochastic generalization of problem solving and control is important for applications, and the theory extends to the stochastic formulation we consider with little difficulty.

To provide a rigorous theory of DP-based learning algorithms in a general setting, we rely on the theory of asynchronous DP. Asynchronous DP does not have to proceed in systematic exhaustive sweeps of the problem's state set as do conventional DP algorithms. Bertsekas [7] and Bertsekas and Tsitsiklis [9] proved general theorems about the convergence of asynchronous DP applied to discrete-time stochastic control problems. They did not, however, relate these results to real-time variants of DP as we do in this article. Watkins [67] also proved a convergence result for a form of asynchronous DP, which he did relate to real-time variants of DP, but his development of the theory is not as extensive as that of Bertsekas and Tsitsiklis. Motivated by real-time applications, Williams and Baird [79] proved additional results about DP algorithms that are asynchronous at a finer grain than those studied by Bertsekas and Tsitsiklis [9]. Although the results of Williams and Baird are relevant to the overall focus of this article, they represent a step beyond what we attempt here.

Korf's [32] LRTA* algorithm is a heuristic search algorithm that caches state evaluations so that search performance improves with repeated trials. Evaluations of the states visited by the problem solver are maintained in a hash table. Each cycle of the algorithm proceeds by expanding the current state by generating all of its neighbors (the states reachable from the current state by the application of some operator) and evaluating them, using previously stored evaluations if they exist in the hash table, and otherwise using an initially given heuristic evaluation function. Assuming the objective is to find a minimum-cost path to a goal state, a score is computed for each neighboring state by adding to its evaluation the cost of the edge to it from the current state. The minimum of the resulting scores becomes the new evaluation for the current state, which is stored in the hash table.[1] Finally, a move is made to this lowest-scoring neighboring state.

LRTA* therefore backs up state evaluations in much the same way as do Samuel's algorithm and DP. In fact, as we shall see in what follows, with a slight caveat, *LRTA* is the deterministic undiscounted specialization of asynchronous DP applied in real-time.* It is therefore closely related to other algorithms based on asynchronous DP. However, Korf's convergence theorem for LRTA* differs significantly from other theorems about the convergence of asynchronous DP. Because Korf developed LRTA* as a kind of heuristic search algorithm, his result addresses convergence to optimal evaluations only for states on optimal solution paths. Although not emphasized by Korf, the implication of this result that is significant from the perspective of DP is that under appropriate conditions LRTA* converges *without the repeated exhaustive expansion of all states required for the convergence of DP algorithms, whether they are conventional or asynchronous, in more general problems.* We prove this in a generalized form in Section 6.

3 Heuristic Search and the Control of Dynamical Systems

We begin by discussing the relationship between heuristic search, real-time heuristic search, and control, as the latter term is used outside of AI.

[1]In Korf's [32] closely related Real-Time A* (RTA*) algorithm, the *second* smallest score is stored. Because LRTA* is more closely related to control and DP than is RTA*, we do not discuss RTA*.

3.1 Heuristic Search and System Control

Heuristic search algorithms, as they have been developed in AI, apply to state-space search problems defined by a set of states, a set of operators that map states to states, an initial state, and a set of goal states. The objective is to find a sequence of operators that maps the initial state to one of the goal states and (possibly) optimizes some measure of cost, or merit, of the solution path. These components constitute a model of some real problem, such as solving a puzzle, proving a theorem, or planning a robot path. The term control as used in the literature on heuristic search and problem solving means the process of deciding what to do next in manipulating a model of the problem in question. Despite some similarities, this is not the meaning of the term control in control theory, where it refers to the process of manipulating the behavior of a physical system in real-time by supplying it with appropriate input signals. In AI, control specifies the formal search process, whereas in control theory, it steers the behavior of a physical system over time. Unlike models manipulated by search algorithms, physical systems cannot be set immediately into arbitrary states and do not suspend activity to await the controller's decisions. Models formalizing system control problems, called *dynamical systems*, take into account the passage of time. In what follows, we shall use the term control as it is used in control theory.

In many applications, a symbolic representation of a sequence of operators is not the final objective of a heuristic search algorithm. The intent may be to execute the operator sequence to generate a time sequence of actual inputs to a physical system. Here the result is the control engineer's form of control, but such a control method differs substantially from the methods addressed by most of control theory. A sequence of control actions produced in this way through heuristic search is an *open-loop control policy*, meaning that it is applied to the system without using information about the system's actual behavior while control is underway, i.e., without feedback. In terms of control theory, heuristic search is a *control design procedure* for producing an open-loop control policy from a system model; the policy is appropriate for the given initial state. Further, under normal circumstances, it is an *off-line* design procedure because it is completed before being used to control the system, i.e, under normal circumstances, the planning phase of the problem-solving process strictly precedes the execution phase.

Open-loop control works fine when all of the following are true: 1) the model used to determine the control policy is a completely accurate model of the physical system, 2) the physical system's initial state can be exactly determined, 3) the physical system is deterministic, and 4) there are no unmodeled disturbances. These conditions hold for some of the problems studied in AI, but they are not true of most realistic control problems. Any uncertainty, either in the behavior of the physical system itself or in the process of modeling the system, implies that *closed-loop control* can produce better performance. Control is closed-loop when each control action depends on current observations of the real system, perhaps together with past observations and other information internal to the controller.

A *closed-loop control policy* (also called a control rule, law, or strategy) is a rule specifying each control action as a function of current, and possibly past, information about the behavior of the controlled system. It closely corresponds to a "universal plan" [51] as discussed, for example, by Chapman [11], Ginsberg [17], and Schoppers [52]. In control theory, a closed-loop control policy usually specifies each control action as a function of the controlled system's current state, not just the current values of observable variables (a distinction whose significance for universal planning is discussed by Chapman [11]). Although closed-loop control is closely associated with negative feedback, which counteracts deviations from desired system behavior, negative feedback control is merely a special case of closed-loop control.

When there is no uncertainty, closed-loop control is not in principle more competent than open-loop control. For a deterministic system with no disturbances, given any closed-loop policy and an initial state, there exists an open-loop policy that produces exactly the same system behavior. It is the open-loop policy generated by running the system, or simulating it with a perfect model, under control of the given closed-loop policy. But this is not true in the stochastic case, or when there are unmodeled disturbances, because the outcome of random and unmodeled events cannot be anticipated in designing an open-loop policy. Note that game-playing systems always use closed-loop control for this reason. The opponent is a disturbance, whose behavior may be inexactly accounted for by minimaxing. A game player always uses the opponent's actual previous moves in determining its next move. For exactly the same reasons, closed-loop control can be better than open-loop control for single-agent problems involving uncertainty. A corollary of this explains the almost universal use of closed-loop control by control engineers: the system model used for designing an acceptable control policy can be significantly less faithful to the actual system when it produces closed-loop instead of open-loop policies. Open-loop control only becomes a practical alternative when the cost of monitoring the controlled system's behavior with sufficient detail is greater than the cost of constructing a model of the system and its disturbances adequate for designing an acceptable open-loop control policy.

Most control theory addresses the problem of designing adequate closed-loop policies off-line when an accurate model of the system to be controlled is available. The off-line design procedure typically yields a computationally efficient method for determining an appropriate control action as a function of the observed system state. If it is possible to design a complete closed-loop policy off-line, as it is in many of the control problems studied by engineers, then it is not necessary to perform any additional re-design, i.e., re-planning, for problem instances differing only in initial state. Changing control objectives, on the other hand, often does require policy re-design.

One can also design closed-loop policies on-line through *repeated* on-line design of open-loop policies. This approach has been called *receding horizon control* [35, 40]. For each current state, an open-loop policy is designed with the current state playing the role of the initial state. The design procedure must terminate within the time constraints imposed by on-line operation. This can be done by designing an optimal finite-horizon open-loop policy, for example, by using a model for searching to a fixed depth from the current state. After applying the first control action specified by the resulting policy, the remainder of the policy is discarded, and the design process is repeated for the next observed state. Despite requiring on-line design, which in AI corresponds to on-line planning through "projection" (i.e., prediction) using a system model, receding horizon control produces a control policy that is reactive to each current system state, i.e., a closed-loop policy. According to this view, then, a closed-loop policy can involve explicit planning through projection, but each planning phase has to complete in a fixed amount of time to retain the system's reactivity to the observed system states. In contrast to methods that design closed-loop policies off-line, receding horizon control easily accommodates changes in control objectives.

3.2 Optimal Control

Perhaps the most familiar control objective is to control a system so that its output matches a reference output or tracks a reference trajectory as closely as possible in the face of disturbances. These are called regulation and tracking problems respectively. In an optimal control problem, on the other hand, the control objective is to extremize some function of the controlled system's behavior, where this function need not be defined in terms of a reference output or trajectory. A typical optimal control problem requires controlling a system to go from an initial state to a goal state via a minimum-cost trajectory. In contrast to tracking problems—where the desired trajectory is part of the problem specification—the trajectory is part of the solution of this optimal control problem. Therefore, optimal control problems such as this are closely related to the problems to which heuristic search algorithms apply.

Specialized solution methods exist for optimal control problems involving linear systems and quadratic cost functions, and methods based on the calculus of variations can yield closed-form solutions for restricted classes of problems. Numerical methods applicable to problems involving nonlinear systems and/or nonquadratic costs include gradient methods as well as DP. Whereas gradient methods for optimal control are closely related to some of the gradient descent methods being studied by connectionists (such as the error-backpropagation algorithm [36, 71]), DP methods are more closely related to heuristic search. Like a heuristic search algorithm, DP is an off-line procedure for designing an optimal control policy. However, unlike a heuristic search algorithm, DP produces an optimal closed-loop policy instead of an open-loop policy for a given initial state.

3.3 Real-Time Heuristic Search

Algorithms for real-time heuristic search as defined by Korf [32] are applicable to state-space search problems in which the underlying model is extended to account for the passage of time. The model thus becomes a dynamical system. Real-time heuristic search algorithms apply to state-space search problems with the additional properties that 1) at each time there is a unique current state of the system being controlled, which is known by the searcher/controller, 2) during each of a sequence of constant-duration time intervals the searcher/controller must commit to a unique action, i.e., choice of operator, and 3) the system changes state at the end of each time interval in a manner depending on its current state and the searcher/controller's most recent action. These factors imply that there is a fixed upper bound on the amount of time the searcher/controller can take in deciding what action to make if that action is to be based on the most up-to-date state information. Thus, whereas a traditional heuristic search algorithm is a *design* procedure for an open-loop policy, a real-time heuristic search algorithm is a *control* algorithm, and it can accommodate the possibility of closed-loop control.

Korf's [32] LRTA* algorithm is a kind of receding horizon control because it is an on-line method for designing a closed-loop policy. However, unlike receding horizon control as studied by control engineers, LRTA* *accumulates* the results of each local design procedure so that the effectiveness of the resulting closed-loop policy tends to improve over time. It stores information from the shallow search forward from each current state by updating the evaluation function by which control decisions are made. Because these updates are the basic steps of DP, we view LRTA* as the result of interleaving the steps of DP with the actual process of control so that control policy design occurs concurrently with control. This approach is advantageous when the control problem is so large and unstructured mathematically that complete control design is not even feasible off-line. This case requires a *partial* closed-loop policy, that is, a policy useful for a subregion of the problem's state space. Designing a partial policy on-line allows actual experience to influence the subregion of the state space where design effort is concentrated. Design effort is not expended for parts of the state space that are not likely to be visited during actual control. Although *in general* it is not possible to design a policy that is *optimal* for a subset of the states unless the design procedure considers the entire state set, this is possible under certain conditions such as those required by Korf's convergence theorem for LRTA*, which we discuss in Section 6.2.

3.4 Adaptive Control

Control theorists use the term adaptive control for cases in which an accurate model of the system to be controlled is not available for designing a policy off-line. Adaptive control algorithms design policies on-line based on information about the control problem that accumulates over time as the controller and system interact. A distinction is sometimes made between adaptive control and learning control, where only the latter takes advantage of *repetitive* control experiences from which information is acquired that is useful over the long term. Although this distinction may be useful for some types of control problems, we think its utility is limited when applied to the kinds of problems and algorithms we consider in this article. According to what we mean by adaptive control in this article, algorithms like LRTA* and Samuel's algorithm [47] are not adaptive algorithms because they assume the existence of an accurate model of the problem being solved. However, because they cause control performance to improve over time by caching the results of experience, they are learning algorithms. Combining methods like these with on-line methods for constructing system models produces learning algorithms applicable to adaptive control problems. In Section 7 we describe several of these algorithms, as well as other algorithms for adaptive control that do not construct explicit system models.

4 Markovian Decision Problems

The basis for our theoretical framework is a class of stochastic optimal control problems called *Markovian decision problems*. This class of problems is the simplest that is general enough to include stochastic versions of the problems to which heuristic search algorithms apply while allowing us to borrow from a well-developed control literature. Markovian decision problems extend the idea of an operator to that of a control action that determines the probabilities that particular states will occur next. More complete descriptions of these problems can be found in many books, such as those by Bertsekas [8] and Ross [46].

A Markovian decision problem is defined in terms of a discrete-time stochastic dynamical system with finite state set $S = \{1, \ldots, n\}$. At each discrete time step, a controller observes the system's current state and generates a control action, or simply an *action*,[2] which is applied as input to the system. If i is the observed state, then the action is selected from a finite set of admissible actions $U(i)$. When the controller generates action $u \in U(i)$, the system's state at the next time step will be j with state-transition probability $p_{ij}(u)$. We further assume that the application of action u in state i incurs an *immediate cost* $c_i(u)$.[3] We do not discuss a significant extension of this formalism in which the controller cannot observe the current state with complete certainty. Although this possibility has been studied extensively and is important in practice, the complexities it introduces are beyond the scope of this article.

[2]In control theory, this is simply called a *control*. We use the term "action" because it is the term commonly used in AI.

[3]To be more general, we can alternatively regard the immediate costs as random numbers depending on states and actions. In this case, if $c_i(u)$ denotes the *expected* immediate cost of the application of action u in state i, the theory discussed below remains unchanged.

When necessary, we refer to states, actions, and immediate costs by the time step at which they occur by using s_t, u_t, and c_t to denote, respectively, the state, action, and immediate cost at time step $t = 0, 1, \ldots$, where $u_t \in U(s_t)$ and $c_t = c_{s_t}(u_t)$. In Section 6 on real-time DP, we elaborate this abstract view by carefully considering the sequence of events that must occur at each time step, but until then it is best to consider each time step as an abstract instant when the controller observes a state, generates an action, and incurs a resulting immediate cost.

A closed-loop policy specifies each action as a function of the observed state. Such a policy is denoted $\mu = [\mu(1), \ldots, \mu(n)]$, where the controller generates action $\mu(i) \in U(i)$ whenever it observes state i. This is a *stationary* policy because it does not change over time. Throughout this paper, when we use the term policy, we always mean a stationary policy. Notice that there are a finite number of policies because both the number of states and the number of actions are finite. For any policy μ, there is a function, f^μ, called the *evaluation function*, or the *cost function*, corresponding to policy μ. It assigns to each state the total cost expected to accumulate over time when the controller uses the given policy starting from the given state. Here, for any policy μ and state i, we define $f^\mu(i)$ to be the *expected total infinite-horizon discounted cost* that will be incurred over time given that the controller uses policy μ and i is the initial state:

$$f^\mu(i) = E_\mu \left[\sum_{t=0}^{\infty} \gamma^t c_t | s_0 = i \right], \tag{1}$$

where γ, $0 \leq \gamma \leq 1$, is a factor used to discount future immediate costs, and E_μ is the expectation assuming the controller always uses policy μ. We refer to $f^\mu(i)$ simply as the *cost* of state i under policy μ. Thus, whereas the *immediate cost* of state i under policy μ is $c_i(\mu(i))$, the *cost* of state i under policy μ is the expected discounted sum of all the immediate costs that will be incurred over the future starting from state i. Theorists study Markovian decision problems with other types of evaluation functions, such as the function giving average cost per-time-step, but we do not consider those formulations here.

The objective of the type of Markovian decision problem we consider is to find a policy that minimizes the cost of each state i as defined by Equation 1. A policy that achieves this objective is an *optimal policy* which, although it depends on γ and is not always unique, we denote $\mu^* = [\mu^*(1), \ldots, \mu^*(n)]$. To each optimal policy corresponds the same evaluation function, which is the *optimal evaluation function*, or *optimal cost function*, denoted f^*; that is, if μ^* is any optimal policy, then $f^{\mu^*} = f^*$. For each state i, $f^*(i)$, the *optimal cost* of state i, is the least possible cost for state i for any policy.

This infinite-horizon discounted version of a Markovian decision problem is the simplest mathematically because discounting ensures that the costs of all states are finite for any policy and, further, that all optimal policies are stationary. The discount factor, γ, determines how strongly expected future costs should influence current control decisions. When $\gamma = 0$, the cost of any state is just the immediate cost of the transition from that state. This is because $0^0 = 1$ in Equation 1 so that $f^\mu(i) = E_\mu[c_0|s_0 = i] = c_i(\mu(i))$. In this case, an optimal policy simply selects actions to minimize the immediate cost for each state, and the optimal evaluation function just gives these minimum immediate costs. Solving a Markovian decision problem with $\gamma = 0$ is much simpler than solving the same problem with $\gamma \neq 0$ because no future consequences of actions have to be considered. As γ increases toward one, future costs become more significant in determining optimal actions, and solution methods generally require more computation.

In the "undiscounted case" when $\gamma = 1$, the cost of a state given by Equation 1 need not be finite, and additional assumptions are required to produce well-defined decision problems. We consider one set of additional assumptions for the undiscounted case because the resulting decision problems are closely related to problems to which heuristic search techniques are usually applied. In these problems, which we call *stochastic optimal path problems* after Bertsekas and Tsitsiklis [9], there is an absorbing set of states, i.e., a set of states that once entered is never left, and the immediate cost associated with applying an action to any of the states in the absorbing set is zero. These assumptions imply that the infinite-horizon evaluation function for any policy taking the system into the absorbing set assigns finite costs to every state even when $\gamma = 1$. This is true because all but a finite number of the immediate costs incurred by such a policy over time must be zero. Additionally, in this case optimal policies remain stationary. The absorbing set of states corresponds to the set of *goal* states in a deterministic optimal path problem, and we call it the *goal set.* However, unlike a state-space search task typically solved via heuristic search, here the objective is to find an optimal closed-loop policy, not just an optimal path from a given initial state.

Researchers in AI studying reinforcement learning often focus on optimal path problems in which all the immediate costs are zero until a goal state is reached, when a "reward" is delivered to the controller and a new trial begins. Problems like this are special kinds of stochastic optimal path problems that allow one to focus on the issue of *delayed reinforcement* [56] in a particularly stark form. Rewards correspond to negative costs in the formalism we are using. In the discounted case when all the rewards are of the same magnitude, an optimal policy produces a shortest path to a rewarding state. Another example of a stochastic optimal path problem receiving attention is identical to this one except that all the non-rewarding immediate costs have the same positive value instead of zero. In this case, an optimal policy produces a shortest path to a goal state in the undiscounted case. These latter problems are examples of minimum-time optimal control problems. We describe an example of one such problem in Section 10 which we use to illustrate some of the algorithms presented.

4.1 The Optimality Equation

To further explain Markovian decision problems and to set the stage for discussing DP, we provide more detail about the relationship between policies and evaluation functions. The evaluation function, f^{μ}, corresponding to policy μ gives the (expected total infinite-horizon discounted) cost for each state, assuming that the controller always uses policy μ. However, μ does not necessarily select actions that lead to the best successor states as evaluated by f^{μ}. In other words, μ is not necessarily a greedy policy with respect to its own evaluation function, f^{μ}.

To define a greedy policy in this stochastic case we use Watkins' [67] "Q" notation because it will play a role in the Q-learning method described in Section 7.5. Let f be a real-valued function of the states; it may be the evaluation function for some policy, a guess for a good evaluation function (such as a heuristic evaluation function in heuristic search), or an arbitrary function. For each state i and action $u \in U(i)$, let

$$Q^f(i,u) = c_i(u) + \gamma \sum_{j \in S} p_{ij}(u) f(j). \qquad (2)$$

$Q^f(i,u)$ is the cost of action u in state i as evaluated by f. It is the sum of the immediate cost and the discounted expected value of the costs of the possible successor states under action u. If the system's state transitions are deterministic, then Equation 2 simplifies to

$$Q^f(i,u) = c_i(u) + \gamma f(j),$$

where j is the successor of state i under action u (i.e, node j is the child of node i along the edge corresponding to operator u). In the deterministic case, one can therefore think of $Q^f(i,u)$ as a summary of the result of a one-ply lookahead from node i along the edge corresponding to operator u as evaluated by f. The stochastic case requires a generalization of this view because many edges correspond to each operator, each having a different probability of being followed. If f is the evaluation function for some policy, $Q^f(i,u)$ gives the cost of generating action u in state i and thereafter following this policy.

Using these "Q-values," a policy μ is greedy with respect to f if for all states i, $\mu(i)$ is an action satisfying

$$Q^f(i,\mu(i)) = \min_{u \in U(i)} Q^f(i,u).$$

Although there can be more than one greedy policy with respect to f if more than one action minimizes the set of Q-values for some state, we let μ^f denote any policy that is greedy with respect to f. Note that it is also true that any policy is greedy with respect to many different evaluation functions.

A key fact underlying all DP methods is that *the only policies that are greedy with respect to their own evaluation functions are optimal policies.* That is, if μ^* is any optimal policy, then its evaluation function is the optimal evaluation function f^*, and $\mu^* = \mu^{f^*}$. This means that for any state i, $\mu^*(i)$ satisfies

$$Q^{f^*}(i, \mu^*(i)) = \min_{u \in U(i)} Q^{f^*}(i, u). \tag{3}$$

Furthermore, any policy that is greedy with respect to f^* is an optimal policy. Thus, if f^* is known, it is possible to define an optimal policy simply by defining it to satisfy Equation 3. Due to the way the Q-values are defined (Equation 2), this generalizes to the stochastic case the fact that an optimal policy is any policy that is best-first with respect to f^* as determined by a one-ply search from each current state. Deeper search is never necessary because f^* already summarizes all the information that such a search would obtain.

Letting $Q^*(i, u) = Q^{f^*}(i, u)$ to simplify notation, a related key fact is that a necessary and sufficient condition for f^* to be the optimal evaluation function is that for each state i it must be true that

$$\begin{aligned} f^*(i) &= \min_{u \in U(i)} Q^*(i, u) \\ &= \min_{u \in U(i)} \left[c_i(u) + \gamma \sum_{j \in S} p_{ij}(u) f^*(j) \right]. \end{aligned} \tag{4}$$

This is one form of the *Bellman Optimality Equation* which can be solved for each $f^*(i)$, $i \in S$, by a DP algorithm. It is a set of n (the number of states) simultaneous nonlinear equations that depends on the dynamical system and the immediate costs underlying the decision problem.

Once f^* has been found, an optimal action for a state i can be determined as follows. The Q-values $Q^{f^*}(i, u)$ for all admissible actions $u \in U(i)$ are determined via Equation 2. In general, this takes $O(mn)$ computational steps, where n is the number of states and m is the number of admissible actions for state i. However, if one knows which of the state-transition probabilities from state i are zero (as one usually does in the deterministic case), then the amount of computation can be much less ($O(m)$ in the deterministic case). Computing these Q-values amounts to a one-ply lookahead search from state i, which requires knowledge of the system's state-transition probabilities. Using these Q-values, an optimal action can be determined via Equation 3, which takes $m - 1$ comparisons. The computational complexity of finding an optimal action using this method is therefore dominated by the complexity of finding f^*, i.e., by the complexity of the DP algorithm.

5 Dynamic Programming

Given a complete and accurate model of a Markovian decision problem in the form of knowledge of the state-transition probabilities, $p_{ij}(u)$, and the immediate costs, $c_i(u)$, for all states i and actions $u \in U(i)$, it is possible—at least in principle—to solve the decision problem off-line by applying one of various well-known DP algorithms. We describe several versions of a basic DP algorithm called *value iteration.*[4] There is another basic DP method called *policy iteration*, but a thorough treatment of real-time algorithms based on policy iteration is beyond the scope of this article, although we briefly discuss them in Section 8. We treat DP as referring only to value iteration unless otherwise noted. As used for solving Markovian decision problems, value iteration is a successive approximation procedure that converges to the optimal evaluation function, f^*. It is a successive approximation method for solving the Bellman Optimality Equation. Its basic operation is "backing up" estimates of the optimal state costs. Several different methods exist for organizing the computations. We first describe the algorithm that backs up costs synchronously.

5.1 Synchronous Dynamic Programming

Let f_k denote the estimate of f^* available at stage k of the computation, where $k = 0, 1, \ldots$. At stage k, $f_k(i)$ is the estimated optimal cost of state i, which we refer to simply as the *stage-k cost* of state i; similarly, we refer to f_k as the *stage-k evaluation function*, even though it may not actually be the evaluation function for any policy. (We use the index k for the stages of a DP computation, whereas we use t to denote the time step of the control problem being solved.) In synchronous DP, f_{k+1} is defined in terms of f_k as follows: for each state i and $k = 0, 1, \ldots$,

$$\begin{aligned} f_{k+1}(i) &= \min_{u \in U(i)} \left[c_i(u) + \gamma \sum_{j \in S} p_{ij}(u) f_k(j) \right] \qquad (5) \\ &= \min_{u \in U(i)} Q^{f_k}(i,u). \end{aligned}$$

We refer to the application of this update equation for state i as *backing up i's cost.* Although backing up costs is a common operation in a variety of search algorithms in AI, there it does not always mean that the backed-up cost is *saved* for future use. Here, however, the backed-up cost is always saved by updating the current evaluation function, which is a permanent data structure.

The iteration defined by Equation 5 is said to be synchronous because no values of f_{k+1}

[4]We should perhaps call this method *cost iteration*, given our problem formulation in terms of cost minimization instead of the equivalent value maximization, but value iteration is the standard term.

appear on the right-hand side of the equation. If we imagine having a separate processor associated with each state, applying Equation 5 for all states i means that each processor backs up the cost of its state at the same time, using the old costs of the other states supplied by the other processors. This process updates all values of f_k simultaneously. Alternatively, a sequential implementation of this iteration would require temporary storage locations so that the stage-$(k+1)$ costs are always computed based on the stage-k costs. The sequential ordering of the backups is irrelevant to the result. If there are n states and m is the largest number of admissible actions for any state, then each iteration, which consists of backing up the cost of each state exactly once, requires at most $O(mn^2)$ operations in the stochastic case and $O(mn)$ operations in the deterministic case. For the large state sets typical of AI problems and many control problems, it is not desirable to try to complete even one iteration, let alone repeat the process until it converges to f^*.

If $\gamma < 1$, repeated synchronous updates produce a sequence of functions that converges to the optimal evaluation function, f^*, for any initial approximation, f_0. Although the cost of a state need not get closer to its optimal cost on each iteration, the *maximum* error between $f_k(i)$ and $f^*(i)$ over all states i must decrease. Mathematically, the synchronous iteration is a contraction mapping with respect to the maximum norm and has f^* as its unique fixed point.

When they converge, synchronous DP and the other off-line value iteration algorithms we discuss below generate sequences of functions that converge to f^*, *but they do not explicitly generate sequences of policies.* To each stage-k evaluation function there corresponds at least one greedy policy, but these policies are never explicitly formed. Ideally, one would wait until the sequence converges to f^* and then form a greedy policy corresponding to f^*, which would be an optimal policy. But this is not possible in practice because value iteration converges asymptotically. Instead, one executes value iteration until it meets a test for approximate convergence and then forms a policy from the resulting function. Unlike value iteration, policy iteration explicitly generates a sequence of policies, and when it converges, it does so after a finite number of iterations because the number of policies is finite. However, policy iteration algorithms have other shortcomings which we discuss in Section 8.

It is important to note that a function in the sequence of evaluation functions generated by value iteration does not have to closely approximate f^* in order for a corresponding greedy policy to be an optimal policy. Indeed, a policy corresponding to the stage-k evaluation function for some k may be optimal long before the algorithm converges to f^*. *But unaided by other computations, value iteration cannot detect when this first happens.* This fact is an important reason that the real-time variants of value iteration we discuss in this article can have major advantages over the off-line variants. The controller makes use of whatever policy is defined by the current evaluation function and so can perform optimally before the evaluation function converges.

Bertsekas [8] and Bertsekas and Tsitsiklis [9] give conditions ensuring convergence of synchronous DP for stochastic optimal path problems in the undiscounted case ($\gamma = 1$). Using their terminology, a policy is *proper* if its use implies a nonzero probability of eventually reaching the goal set starting from any state. Using a proper policy also implies that the goal set will be reached eventually from any state with probability one. The existence of a proper policy is the generalization to the stochastic case of the existence of a path from any initial state to the goal set.

Synchronous DP converges to f^* in undiscounted stochastic optimal path problems under the following conditions:

1. the initial cost of every goal state is zero,
2. there is at least one proper policy, and
3. all policies that are not proper incur infinite cost for at least one state.

The first condition makes intuitive sense, and the third condition ensures that every optimal policy is proper, i.e, it rules out the possibility that a least-cost path exists that never reaches the goal set. This third condition is true if all immediate costs for transitions from non-goal states are positive, i.e, $c_i(u) > 0$ for all non-goal states i and actions $u \in U(i)$.[5] In the deterministic case, the latter two conditions are satisfied if there exists at least one solution path from each initial state and the sum of the immediate costs in every loop is positive.

5.2 Gauss-Seidel Dynamic Programming

Gauss-Seidel DP differs from the synchronous version in that the costs are backed up one state at a time in a sequential "sweep" of all the states, with the computation for each state using the most recent costs of the other states. If we assume that the states are numbered in order, as we have here, and that each sweep proceeds in this order, then the result of each iteration of Gauss-Seidel DP can be written as follows: for each state i and each $k = 0, 1, \ldots$,

$$\begin{aligned} f_{k+1}(i) &= \min_{u \in U(i)} \left[c_i(u) + \gamma \sum_{j \in S} p_{ij}(u) f(j) \right] \\ &= \min_{u \in U(i)} Q^f(i, u). \end{aligned}$$

where

$$f(j) = \begin{cases} f_{k+1}(j) & \text{if } j < i \\ f_k(j) & \text{otherwise.} \end{cases}$$

[5]This assumption of positive immediate costs can be weakened to nonnegativity, i.e., $c_i(u) \geq 0$ for all $i \in S$ and $u \in U(i)$, if there exists at least one *optimal* proper policy [9].

Unlike synchronous DP, the order in which the states' costs are backed up influences the computation. Nevertheless, Gauss-Seidel DP converges to f^* under the same conditions under which synchronous DP converges. When $\gamma < 1$, repeated Gauss-Seidel sweeps produce a sequence of functions that converges to f^*. The cost estimate of any individual state may not improve on a sweep, but each sweep decreases the maximum error in the costs over all the states. For stochastic optimal path problems with no discounting, the conditions described above that ensure convergence of synchronous DP also ensure convergence of Gauss-Seidel DP [9]. Because each cost backup uses the latest costs of the other states, Gauss-Seidel DP tends to converge faster than synchronous DP. Furthermore, it should be clear that some state orderings will produce faster convergence than others, depending on the problem. For example, in optimal path problems, sweeping from goal states backwards along likely optimal paths may lead to faster convergence than sweeping in the forward direction.

Although Gauss-Seidel DP is not one of the algorithms of direct interest in this article, we used it to solve the example problem described in Section 11, and it serves as a bridge between the usual synchronous version of DP and the asynchronous version discussed next.

5.3 Asynchronous Dynamic Programming

Asynchronous DP is similar to Gauss-Seidel DP in that it does not back up state costs simultaneously. However, it is not organized in terms of systematic successive sweeps of the state set. As proposed by Bertsekas [7] and further developed by Bertsekas and Tsitsiklis [9], asynchronous DP is suitable for multi-processor systems with communication time delays and without a common clock. For each state $i \in S$ there is a separate processor dedicated to backing up the cost of state i (more generally, each processor may be responsible for a number of states). The times at which each processor backs up the cost of its state can be different for each processor. To back up the cost of its state, each processor uses the costs for other states that are available to it when it "awakens" to perform a backup. Multi-processor implementations have obvious utility in speeding up DP and thus have practical significance for all the algorithms we discuss below (see Lemmon [37]). However, our theoretical interest in asynchronous DP lies in that fact that it eliminates the necessity to back up state costs in any systematically organized fashion. We therefore describe a special case of asynchronous DP that is more suitable for our purposes.

Although in the full asynchronous model, the notion of discrete computational stages does not apply because a processor can awaken at any of a continuum of times, we use a notion of stage because it will facilitate our discussion of real-time DP in the next section. As in the other forms of DP, let f_k denote the estimate of f^* available at stage k of the computation, where $k = 0, 1, \ldots$. At each stage k, the costs of a *subset* of the states are backed up synchronously, and the costs remain unchanged for the other states. The subset of states whose costs are backed up changes from stage to stage, and the choice of these subsets determines the precise nature of the algorithm. For each $k = 0, 1, \ldots$, if $S_k \subseteq S$ is the set of states whose costs are backed up at stage k, then f_{k+1} is computed as follows:

$$f_{k+1}(i) = \begin{cases} \min_{u \in U(i)} Q^{f_k}(i, u) & \text{if } i \in S_k \\ f_k(i) & \text{otherwise.} \end{cases} \tag{6}$$

According to this algorithm, then, f_{k+1} may differ from f_k on one state, on many states, or possibly none, depending on S_k. Further, the costs of some states may be backed up several times before the costs of others are backed up once. Asynchronous DP clearly includes the synchronous and Gauss-Seidel algorithms as special cases: synchronous DP results if $S_k = S$ for each k; Gauss-Seidel DP results when each S_k consists of a single state and the collection of S_ks is defined to implement successive sweeps of the entire state set (e.g., $S_0 = \{1\}$, $S_1 = \{2\}$, …, $S_{n-1} = \{n\}$, $S_n = \{1\}$, $S_{n+1} = \{2\}$, …).

Discounted asynchronous DP converges to f^* provided that the cost of each state is backed up infinitely often, i.e., provided that each state is contained in an infinite number of the subsets S_k, $k = 0, 1, \ldots$. In practice, this simply means that whatever strategy is used to select states whose costs are to be backed up, no state should ever be completely barred from selection in the future.

It is important to realize that a single backup of a state's cost in asynchronous DP *does not necessarily improve it as an estimate of the state's optimal cost*; it may in fact make it worse. However, with repeated backups, the cost of each state converges to its optimal cost. Further, as in Gauss-Seidel DP, the order in which states' costs are backed up can influence the rate of convergence in a problem-dependent way. This fact underlies the utility of various strategies for "teaching" algorithms based on asynchronous DP by supplying experience dictating selected orderings of the backups (e.g., Lin [38], Utgoff and Clouse [66], and Whitehead [77]).

In the undiscounted case ($\gamma = 1$), additional assumptions are necessary beyond the assumption that the cost of each state is backed up infinitely often. The simplest conditions, and also the most restrictive, under which Bertsekas and Tsitsiklis [9] prove that asynchronous DP converges to f^* in undiscounted stochastic optimal path problems are the following:

1. the initial cost for every goal state is zero, and
2. *every* policy is proper.

The second condition means that no matter what actions are generated, there is a nonzero probability of eventually reaching a goal state. This is equivalent to the strong assumption that a goal state will be reached with probability one independently of the controller's policy.

In the deterministic case, this means that every path leads to a goal state. Although this assumption might be justifiable in some problems, it does not apply to most of the optimal path problems one would wish to consider.

However, asynchronous DP also converges for undiscounted stochastic optimal path problems under the following more satisfactory conditions:

1. the initial cost of every goal state is zero,
2. there is at least one proper policy, and
3. all immediate costs incurred by transitions from non-goal states are positive, i.e., $c_i(u) > 0$ for all non-goal states i and actions $u \in U(i)$.

Except for condition 3, these conditions are identical to a set of conditions enumerated in Section 5.1 guaranteeing convergence of synchronous DP in undiscounted stochastic optimal path problems. For the synchronous case, condition 3 is the weaker condition that all improper policies incur infinite cost for at least one state. When there exist improper policies, convergence of asynchronous DP requires the stronger condition of positive immediate costs. This is illustrated by a simple example in Appendix A. Because a proof of convergence of asynchronous DP in undiscounted stochastic optimal path problems under these conditions does not, to the best of our knowledge, appear in the literature, we provide one in Appendix B. Our proof is a straightforward use of the general machinery developed by Bertsekas and Tsitsiklis [9].

6 Dynamic Programming in Real Time

The DP algorithms described above are off-line methods for solving Markovian decision problems. Although they successively approximate the optimal evaluation function through a sequence of computational stages, these stages are not related to the time steps of the decision problem being solved. Here we consider algorithms in which the controller performs DP on-line during control, with the computational steps of an off-line DP algorithm *interleaved* with the generation of actions. Throughout this section we assume that there is a complete and accurate model of the decision problem, the case Sutton [59] discusses in relation to planning in AI. The utility of performing DP on-line when there is a complete and accurate model of the decision problem lies in the fact that the state and admissible action sets may be so large that it is impractical to run a DP algorithm off-line until it converges to the complete optimal evaluation function. Under these conditions, it still might be practical to form a useful evaluation function for just some of the problem's states. Interleaving DP with the generation of actions allows these states to depend on the states actually visited by the controller so that they are likely to be most relevant to future performance of the control system.

Generalizing Korf's [32] convergence theorem for LRTA*, we show that under certain conditions this process can result in evaluation functions that are in fact *optimal* for a relevant subset of states. In Section 6.3, we describe in detail how LRTA* is related to DP. In Section 7, we discuss the adaptive case, in which we do not assume a complete and accurate model of the decision problem. In this case, interleaving asynchronous DP stages with control time steps produces adaptive algorithms with other computational advantages.

6.1 Interleaving DP with Control

To describe various ways of interleaving the stages of DP with control steps, we regard the controller as performing the computational stages of DP as well as generating actions that influence the system underlying the decision problem. The controller can perform a certain portion of a DP algorithm between the times at which it is required to generate actions, where the amount of DP it can perform depends on its computational resources. Accordingly, we elaborate the abstract discrete-time formulation of a Markovian decision problem presented in Section 4 by expanding each time step into an interval of real time of finite duration (Figure 1). At the beginning of the interval corresponding to time step t, which we call interval t, the controller observes state s_t; it must generate an action $u_t \in U(s_t)$ before the end of the interval. At the beginning of the next interval, interval $t+1$, the controller observes the new state, s_{t+1}, which has been determined by s_t, u_t, and the state-transition probabilities of the system underlying the decision problem. For simplicity, and without significant loss of generality, we assume that each action is generated at the end of its interval and that the resulting new state is available to the controller after a negligible delay.

Interleaving DP with on-line control means that during the time intervals of the decision problem, the controller executes some finite number of stages of asynchronous DP as defined by Equation 6. The results of asynchronous DP accumulate over time, as each stage begins with the evaluation function produced by the previous stage. Control decisions are made based on the most recent evaluation function. Equivalently, we may view this as executing asynchronous DP concurrently with the process of controlling the system, with the controller having available the most up-to-date evaluation function.

The asynchronous version of DP is appropriate for this role because of the flexibility with which its stages can be defined. Not only does asynchronous DP encompass the other DP algorithms discussed above as special cases, its stages can be defined on-line in a way that is responsive to the observed behavior of the system being controlled. The states whose costs are backed up in a stage can be selected so that the information gained is useful for the current control task. Additionally, one can usually define the stages of asynchronous DP so that they require as little computation as needed to meet the time constraints imposed by on-line control.

Although it is not necessary to interleave DP stages with control steps in the strict sense that a stage has to complete by the end of a control time interval, for simplicity, and without significant loss of generality, we assume that no DP stage continues from one time interval to the next, i.e., that no control action is generated while a stage is still being computed.

To make this specific, suppose that at the beginning of each interval t, $t = 0, 1, \ldots$, the controller observes state s_t and has available the estimate of the optimal evaluation function produced by all the stages of asynchronous DP already executed. Suppose this estimate is f_{k_t}, where $k_t \geq 0$ is the total number of stages executed up to the beginning of interval t. By the end of interval t, the controller executes n_t additional stages of asynchronous DP, resulting in updating f_{k_t} to $f_{k_{t+1}}$, where $k_{t+1} = k_t + n_t$. The controller then generates an action $u_t \in U(s_t)$ based on $f_{k_{t+1}}$, which is input to the system, yielding the immediate cost $c_{s_t}(u_t)$. The system's state changes to s_{t+1}, and the process repeats for interval $t+1$. This general scheme for interleaving asynchronous DP with on-line control is illustrated in Figure 1.

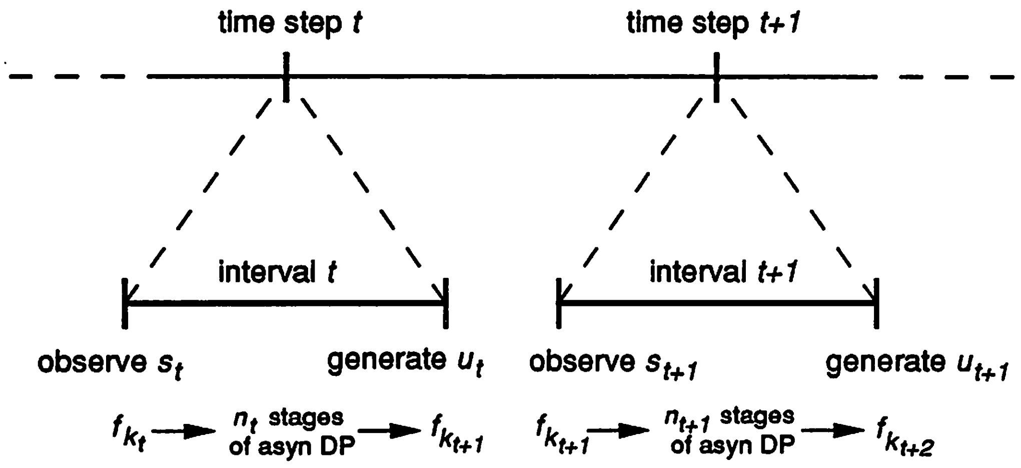

Figure 1: Interleaving Asynchronous DP with On-line Control. Each abstract time step t of a Markovian decision problem is expanded into time interval t. At the beginning of interval t, f_{k_t} is the current evaluation function, and the controller observes state s_t. The controller executes n_t stages of asynchronous DP during the interval to produce evaluation function $f_{k_{t+1}}$, on whose basis action u_t is generated at the interval's end. We assume that the controller observes the next state, s_{t+1}, some negligible time after generating u_t, shown as the gap between the intervals t and $t+1$.

According to this notation, $k_0 = 0$, and since $k_{t+1} = k_t + n_t$ for $t = 0, 1, \ldots$, $k_t =$

$n_0 + n_1 + \ldots + n_{t-1}$. At the beginning of interval t, the costs of the states in the set $\cup_{k=0}^{k_t} S_k$ have been backed up at least once, where S_k is the set of states whose costs are backed up at stage k of asynchronous DP as defined by Equation 6. We will find it useful to refer to the set of states whose costs are backed up during interval t. Denoting this set B_t, it is given by

$$B_t = \cup_{k=k_t+1}^{k_t+n_t} S_k,$$

for $t = 0, 1, \ldots$. Notice that if any state in B_t is an element of more than one of the sets S_k in this union, then its cost is backed up more than once during interval t.

6.2 Real-Time DP

Mathematically, then, an asynchronous DP algorithm defined by the sequence of sets S_k, $k = 0, 1, \ldots$, is interleaved with on-line control by specifying the numbers n_t, $t = 0, 1, \ldots$. Because this does not alter the mathematical properties of the asynchronous DP algorithm, the conditions for its convergence described in Subsection 5.3 continue to apply. We use the term *real-time DP* to refer to cases in which the stages of asynchronous DP and the time steps of on-line control are not only interleaved, but influence one another in the following ways. First, in real-time DP, the controller always follows a policy that is greedy with respect to the most recent estimate of f^*. Because action u_t is generated at the end of interval t, this estimate is $f_{k_{t+1}}$. Moreover, any ties in selecting these actions must be resolved randomly, or in some other way that ensures the continuing selection of all the greedy actions. Second, in real-time DP, B_t, the set of states whose costs are backed up during interval t, always contains s_t. In the simplest case of real-time DP, the cost of *only* s_t is backed up at each time step. This is the case in which $n_t = 1$ and $B_t = S_t = \{s_t\}$ for all t. More generally, in addition to s_t, B_t can contain *any* states, such as those generated by any type of off-line lookahead search. For example, B_t might consist of the states generated by an exhaustive off-line search from s_t forward to some fixed search depth, or it might consist of the states generated by some form of best-first search according to the most recent estimate of f^*.

Although these choices can greatly influence the rate at which real-time DP converges to f^*, they have no influence on whether or not it converges. *Because it is a form of asynchronous DP, real-time DP converges to the optimal evaluation function under the conditions ensuring convergence of asynchronous DP.* In the discounted case, the only condition necessary for convergence is that no state is ever completely ruled out for having its cost backed up. Because real-time DP always backs up the cost of the current state, one way to achieve this is to make sure that the controller always continues to visit each state. There are several different approaches to ensuring this.

One approach is to assume, as is often done in the engineering literature, that the Markov process resulting from the use of any policy is ergodic. This means that all states always retain a nonzero probability of being visited no matter what actions are generated. Discounted real-time DP converges under this assumption. However, this assumption does not allow proper subsets of states to be absorbing, which rules out nontrivial stochastic optimal path problems because their goal sets must be absorbing.

A second way to ensure that each state is visited infinitely often is to use multiple *trials*. A trial consists of a time interval of finite duration during which real-time DP is performed while the controller and system interact. After this interval, the system's state is set to a new starting state, and a new trial begins. This amounts to performing real-time DP where the current state is sometimes determined by the controller/system interaction and sometimes by the intervention of another process which starts a new trial. The end of a trial interrupts the real-time DP algorithm so that the cost of the last state in a trial is not influenced by the cost of the starting state of the next trial. This prevents the state transitions caused by the process that initiates and terminates trials from influencing the evaluation function. If this process initiates trials by selecting states so that every state will be selected infinitely often in an infinite series of trials, then obviously every state will be visited infinitely often—if only at the start of an infinite number of trials. A simple way to accomplish this is to start each trial with a randomly selected state, where each state has a nonzero probability of being selected. By *trial-based* real-time DP we mean real-time DP involving trials initiated so that every state will be a start state infinitely often in an infinite series of trials. For the discounted case, trial-based real-time DP is an example of a convergent asynchronous DP algorithm, and thus it always converges to f^*.[6]

It is natural to use trial-based real-time DP in undiscounted stochastic optimal path problems, letting each trial continue until the system reaches a goal state.[7] During each trial, the controller's actions determine which states are visited and, hence, which states' costs are backed up. Consequently, trial-based real-time DP solves undiscounted stochastic optimal path problems under the conditions enumerated in Section 5.3 guaranteeing convergence of asynchronous DP for these problems.

Real-time DP is more interesting if we relax the requirement that it should yield a

[6]This is the f^* for the Markovian decision problem on which the multiple trials are imposed. It does not reflect characteristics of the process that imposes these trials because the cost of the last state in each trial is not backed up at the trial's end. If real-time DP were not interrupted in this way, then the trial-based computation would be equivalent to applying real-time DP without using multiple trials to *some other Markovian decision problem based on a different dynamical system*. The trials would be part of this other system's behavior, and the optimal evaluation function for this different Markovian decision problem would not necessarily equal f^* of the original problem. This is why we consider using multiple trials to be an aspect of an algorithm instead of a property of a control problem. Obviously, however, trial-based algorithms are not applicable to control problems in which it is not possible to set the system state to selected start states. This requires a controller that may not exist in principle or that is too difficult to design in practice.

[7]It is also possible to let a trial "time out" after some number of time steps. With simple modifications, the theory can be extended to this case, but we do not address this extension here.

complete optimal evaluation function, from which one can determine a *complete* optimal policy. Here we rely on Korf's [32] insights as embodied in LRTA*, but we generalize them to trial-based real-time DP applied to stochastic optimal path problems. Consider a trial-based approach to solving undiscounted stochastic optimal path problems in which there is a designated subset of *start states* with which trials always start. Further, we restrict the task's objective to finding the optimal costs only for states that can be reached from a start state when following an optimal policy. Clearly, if the set of start states contains every nongoal state, then this is the undiscounted stochastic optimal path problem defined above. When there are states that are neither goal nor start states, it is possible that the costs of some of them will never be backed up, or will be backed up only a finite number of times. One cannot be assured that the costs of such states will converge to the correct values. However, one can give conditions ensuring that such states cannot lie on optimal trajectories from start states. We state this as a theorem, which is a generalization of Korf's [32] convergence theorem for LRTA*, and our proof given in Appendix C generalizes his proof by invoking results for asynchronous DP:

Theorem (Trial-Based Real-Time DP): In undiscounted stochastic optimal path problems, trial-based real-time DP, with the initial state of each trial restricted to a set of start states, ensures that the costs of all states that can be reached from any start state using an optimal policy converge with probability one to their optimal costs under the following conditions: 1) the initial cost of every goal state is zero, 2) there is at least one proper policy, 3) all immediate costs incurred by transitions from non-goal states are positive, i.e., $c_i(u) > 0$ for all non-goal states i and actions $u \in U(i)$, and 4) the initial costs of all states are non-overestimating, i.e., $f_0(i) \leq f^*(i)$ for all states $i \in S$.

Trial-based real-time DP can therefore yield an evaluation function that is optimal only for the states whose costs are needed to define an optimal policy in a stochastic optimal path problem. When the set of start states is a proper subset of the non-goal states, the controller's policy always converges to an optimal policy without the requirement that it back up the costs of all states infinitely often. More importantly in practice, the costs of some states might not have to be backed up at all and may not even have to be represented.[8]

The significance of this result, as well as the other convergence results for real-time DP, is that when state and action sets are too large to realistically permit the completion of off-line DP, real-time DP can eventually yield optimal solutions even with limited computational resources. Whereas with conventional off-line DP, computational limits postpone the *commencement* of control, with real-time DP, they postpone its *convergence* to optimal-

[8]If trials are allowed to time out before a goal state is reached, it is possible to eliminate the requirement that there exist at least one proper policy. Timing out prevents getting stuck in fruitless cycles, and the time-out period can be extended systematically to ensure that it becomes long enough to let all the optimal paths be followed without interruption.

ity. Although performance is not guaranteed to improve on each time step, it is guaranteed to improve over many time steps. When the policy does improve significantly long before the algorithm converges, real-time DP allows the controller to automatically take advantage of this improvement. There is no need to wait until a conventional off-line DP algorithm satisfies a pre-determined convergence criterion. Moreover, because the stages of real-time DP are guided by the controller's actual experience, computational resources are focused on regions of the state space where they are likely to be most beneficial.

Although in both discounted and undiscounted problems, the eventual convergence of real-time DP does not depend critically on the choice of states whose costs are backed up at each time interval, judicious selection of these states can accelerate convergence. Sophisticated exploration strategies can be implemented by selecting these states based on prior knowledge and the information contained in the current evaluation function. For example, in a trial-based approach to a stochastic optimal path problem, guided exploration can reduce the expected trial duration by helping the controller find goal states. It also makes sense for real-time DP to back up the costs of states whose current costs are not yet accurate estimates of their optimal costs but whose successor states do have accurate current costs. Techniques for "teaching" DP-based learning systems by suggesting certain back ups over others (refs. [38, 77, 66]) rely on the fact that the order in which the costs of states are backed up can influence the rate of convergence of asynchronous DP, whether applied off- or on-line. Exploration such as this—whose objective is to facilitate finding an optimal policy when there is a complete model of the decision problem—must be distinguished from exploration designed to facilitate learning a model of the decision problem in the adaptive case. We discuss this latter objective for exploration in Section 7.2.

6.3 Real-Time DP and LRTA*

Real-time DP extends Korf's [32] LRTA* algorithm in two ways: it generalizes LRTA* to stochastic problems, and it can back up the costs of many states at each time step of on-line control, whereas LRTA* backs up only the cost of the current state. Using our notation, the simplest form of LRTA* operates as follows: at each time interval t, the controller observes state s_t and backs up its cost by setting $f_{t+1}(s_t)$ to the minimum of the values $c_{s_t}(u) + \gamma f_t(j)$ for all actions $u \in U(s_t)$, where j is s_t's successor under action u and $f_t(j)$ is j's current cost.[9] The costs of all the other states remain the same. The controller then inputs this minimizing action to the system, observes s_{t+1}, and repeats the process.

This form of LRTA* is almost the special case of real-time DP as applied to a deterministic problem in which $B_t = \{s_t\}$ for all $t = 0, 1, \ldots$. To be exactly this special case, the controller would have to generate the action that is optimal with respect to f_{t+1} instead of f_t, i.e., the

[9]Note that because $n_t = 1$ for all t, k_t always equals t.

action $u \in U(s_t)$ that minimizes $c_{s_t}(u) + \gamma f_{t+1}(j)$, where j is s_t's successor under action u. This is usually an inconsequential difference because $f_t(j)$ can differ from $f_{t+1}(j)$ only when $j = s_t$, i.e., when s_t is its own successor. LRTA* saves computation by requiring only one minimization at each time step: the minimization required to perform the backup also gives the action. However, in the general case, when real-time DP backs up more than one state's cost during each time interval, it makes sense to use the latest approximation of f^* to select an action.

An extended form of LRTA* has other similarities with real-time DP. In all of his discussion, Korf [32] assumes that the cost of a state may be *augmented by lookahead search.* This means that instead of using the current costs $f_t(j)$ of s_t's successor states, LTRA* performs an off-line forward search from s_t to a depth determined by the available computational resources. It applies the current evaluation function f_t to the nodes at the frontier and then backs up these costs to s_t's immediate successors. This is done (roughly) by setting the backed-up cost of each state generated in the forward search to the minimum of the costs of its successors (Korf's "minimin" procedure). These backed-up costs of the successor states are then used to update $f_t(s_t)$, as described above, *but neither these costs nor the backed-up costs of the states generated in the forward search are saved.* Despite the fact that backed-up costs for many states have been computed, the new evaluation function, f_{t+1}, differs from the old only for s_t. In the absence of space constraints dictating that it is impractical to save the backed-up costs of all of the states generated during LRTA* or real-time DP, it makes sense to store backed-up costs for as many states as possible, especially when the controller will experience multiple trials with different starting states.

The corresponding real-time DP algorithm, on the other hand, would save all of these backed-up costs in $f_{k_{t+1}}$. In this algorithm, stages of asynchronous DP executed during time interval t back up the costs of all the states expanded in the forward search from s_t. Specifically, saving the backed-up costs produced by Korf's minimin procedure corresponds to executing a number of stages of asynchronous DP equal to one less than the depth of the forward search tree. The first stage synchronously backs up the costs of all the immediate predecessors of states on the frontier of the search tree (and thus uses the current costs of the frontier states), the second stage backs up the costs of the states that are the immediate predecessors of these states, etc. Then one additional stage of asynchronous DP to back up the cost of s_t completes the computation of $f_{k_{t+1}}$. Not only does this also apply in the stochastic case, it suggests that other stages of asynchronous DP might be useful as well. These stages might back up the costs of states not in the forward search tree, or they might back up states in this tree more than once. For example, noting that in general the forward search might generate a graph with cycles instead of a tree, multiple backups of the costs of these states can further improve the information contained in $f_{k_{t+1}}$. All of these possibilities are basically different instances of real-time DP and thus converge under the general conditions described in Section 6.2.

With repeated trials, the information accumulating in the developing estimate of the optimal evaluation function improves control performance. Consequently, LRTA* and real-time DP are indeed learning algorithms, as suggested by the name chosen by Korf. However, they do not directly apply to *adaptive* control problems as this term in used in control theory. In the next section we discuss adaptive control and describe a number of novel methods that rely on real-time DP or closely related algorithms.

7 Adaptive Control

All of the DP algorithms described above—synchronous, Gauss-Seidel, asynchronous, and real-time—require prior knowledge of the system underlying the Markovian decision problem. That is, they require knowledge of the state-transition probabilities, $p_{ij}(u)$, for all states i, j, and all actions $u \in U(i)$, and they require knowledge of the immediate costs $c_i(u)$ for all states i and actions $u \in U(i)$. If the system is deterministic, this means that one must know the successor states and the immediate costs for all the admissible actions for every state. The problem of finding, or approximating, an optimal policy when this knowledge is not available is the *adaptive* variant of a Markovian decision problem. Several different formulations of the adaptive problem have been studied. The survey by Kumar [33] provides an overview of this large literature and conveys the subtlety of the issues as well as the sophistication of the existing theoretical results. Our purpose here is to discuss methods that are related to real-time DP and to provide an accurate, though non-rigorous, discussion of some of the important issues.

Solution methods for adaptive Markovian decision problems fall into two main classes. Bayesian methods rest on an assumption of a known *a priori* probability distribution over the class of possible stochastic dynamical systems. As observations accumulate over time, this distribution is successively revised via Bayes' rule, and at each time step, an action is selected by using DP to find a policy that minimizes the expected cost over the set of possible systems as well as over time. To do this, DP is applied to a decision problem involving a system whose state consists of an entire *posterior* distribution over the set of possible systems. Although this approach yields insights about the optimal way to combine control with exploration, the computations required can be prohibitive in all but the simplest cases. Non-Bayesian approaches, in contrast, do not involve manipulating probability distributions over sets of possible systems. Instead of attempting to define the best action at each time step on the basis of such a distribution, a non-Bayesian method attempts to arrive at an optimal policy *asymptotically* for *any* system within some pre-specified class of systems. Consequently, at each time step the action may not be optimal on the basis of prior assumptions and accumulated observations, but in the limit as experience accumulates, the policy should approach an optimal policy. Here we restrict attention to non-Bayesian methods.

7.1 Indirect and Direct Adaptive Methods

Within the class of non-Bayesian methods for adaptive control, it is usual to distinguish between *indirect* and *direct* methods. An indirect method relies on the on-line formation of an explicit model of the dynamical system being controlled as well as estimates of the immediate costs, if the latter are also unknown. A policy is determined by using some control design procedure with the current model substituted for the true model of the system. This is based on what control theorists call the *certainty equivalence* principle [8]. Under specialized conditions, it makes sense to always generate the action given by the policy that is optimal for the current model, called the *certainty equivalence optimal policy*. (As we discuss in Section 7.2, however, this is not true in general.) Direct adaptive control methods, on the other hand, determine a policy without forming an explicit system model.

More specifically, an indirect method forms a system model on-line by using observations obtained during interaction with the system. This is done by a *system identification* algorithm that estimates parameters whose values determine the current model at any time step. For a Markovian decision problem, for example, at any time step t, the model consists of current estimates of the state-transition probabilities, $p^t_{ij}(u)$, for all states i, j, and all admissible actions $u \in U(i)$, and current estimates of immediate costs $c^t_i(u)$ for all states i and actions $u \in U(i)$. These estimates are then treated as if they were the actual state-transition probabilities and immediate costs, and a DP algorithm is used to obtain the evaluation function that would be optimal if this were true. We call this the *certainty equivalence optimal evaluation function*, from which the action u_t is determined. As we discuss in Section 7.2, however, to allow for exploration u_t should not always be the greedy action with respect to this evaluation function. At the next time step, the system identification algorithm updates the model parameters based on the new observations, DP is applied again, and the process repeats.

Indirect methods, therefore, require performing control design on-line because the system model is being updated on-line. On-line control design consists of *repeatedly* executing a design procedure that is usually executed *once* off-line in non-adaptive problems. Consequently, the computational cost of the design procedure is a significant factor in determining the feasibility of an indirect adaptive control method. In some problems (e.g., adaptive linear regulation and tracking problems), the design procedure is not computationally complex, whereas in other problems, its complexity is a serious impediment to applying an indirect method. For example, most indirect methods proposed for solving adaptive Markovian decision problems require running a conventional DP algorithm to a satisfactory degree of convergence *at each time step* during adaptive control. Even in the most optimistic case in which synchronous or Gauss-Seidel DP converges in one iteration, for a system with n states and m admissible actions for each state, at each time step of control $O(mn^2)$ operations would be required in the stochastic case and $O(mn)$ operations would be required in the deterministic case. Although parallel computation can allow each iteration to be performed more rapidly (see Lemmon [37]), this complexity seriously limits the utility of these methods.

In contrast to indirect methods, direct methods bypass the explicit identification of the system being controlled. They directly estimate either a suitable policy or information other than a system model, such as an evaluation function, from which a suitable policy can be determined. Direct methods have the advantage of not requiring repeated application of a control design procedure. However, as usually considered in adaptive control (e.g., Goodwin and Sin [19]), direct methods are possible only when the control design procedure they replace is sufficiently simple computationally. In these cases, one can express the adaptive changes in the policy in terms of the training information, eliminating the policy's dependence on an explicit system model. Unfortunately, the control design procedure for general Markovian decision problems is some form of DP, whose complex dependence on a system model would seem to preclude the possibility of employing direct methods for these problems. Consequently, most approaches to solving adaptive Markovian decision problems are indirect. However, direct methods have also been studied. Narendra and Wheeler [76] developed a direct method that completely bypasses DP and effectively adjusts a policy based on estimates of the costs of states obtained over extended periods of control. Other direct methods, some of which we describe below, do utilize the principles of DP but do so without forming explicit system models. They estimate the optimal evaluation function, f^*, by means of DP-based operations that do not rely on a system model. For these methods, it still makes sense to use the term certainty equivalence optimal policy to refer to a policy that is greedy with respect to the current estimate of f^*.

Although systematic studies of the relative advantages of indirect and direct methods appear to be lacking, it is clear that the results would be highly dependent on the type of adaptive control problem under study—whether it involves tracking or optimal control—and on its specific features determining the relative complexity of system identification compared to the basic control problem.[10] According to conventional wisdom, in adaptive regulation and tracking problems for linear systems, it makes little difference whether the method is indirect or direct. For general Markovian decision problems, where the control design procedure is much more complex, only the following seems clear: Indirect methods in which DP is used to find a certainty equivalence optimal policy *every time the system model is updated* scale extremely poorly with increasing numbers of states and actions. This is the type of indirect method that has received the most attention in the literature on adaptive Markovian decision problems. We discuss this type of method as the *generic indirect method* in Section 7.3.

Other methods, based on real-time DP and Watkins' [67] Q-learning algorithm, discussed in Sections 7.4 and 7.5, appear to be much more practical because they interleave the stages of DP with the time steps of control. Barto and Singh [5] discuss some of these issues and describe comparative results for a simple adaptive Markovian decision problem. Jalali and

[10]Gullapalli [20] argues that in some control problems, system identification is harder than the control problem itself, so that direct methods can be more efficient. Barto and Singh [5] discuss related issues.

Ferguson [24] also discuss the computational efficiency of interleaving the stages of DP with control steps. In Section 11 we present simulation results for adaptive methods based on real-time DP and Q-learning on a problem having such a large number of states that the generic indirect method is decidedly impractical.

7.2 Exploration

If one can guarantee that the sequence of system models generated by an indirect adaptive method converges to the true system, then obviously the sequence of certainty equivalence optimal policies also converges to an optimal policy. However, such a guarantee is difficult to provide if identification and control are to be conducted together during interaction with the system. Even if the true system is within the class of models capable of being identified (which is usually assumed), correct identification requires continuing exploration that is at odds with improving control. Exploring to identify an unknown system is different than exploring to facilitate discovering an optimal policy when the system is known as discussed above in Section 6.2.

The conflict between identification and control is a central issue in adaptive optimal control. How does one combine exploration sufficient to achieve model convergence with the objective of eventually following an optimal policy? Further, what kind of performance is achieved *before* the resulting sequence of policies converges? Although they do not construct a system model, direct approaches to adaptive optimal control also require exploration and involve these same issues. If they are to be proved to converge, adaptive optimal control algorithms require mechanisms for resolving these problems, but no mechanism is universally favored. Some of the approaches for which rigorous theoretical results are available are reviewed by Kumar [33], and a variety of more heuristic approaches have been studied by Barto and Singh [5], Kaelbling [27], Moore [44], Schmidhuber [49], Sutton [58], Watkins [67], and Thrun and Möller [65].

In the following subsections, we describe several non-Bayesian approaches to solving adaptive Markovian decision problems. Although these approaches can form the basis of algorithms that can be proved to converge to optimal policies, we do not describe the exploration mechanisms with enough rigor for developing the theory in this direction. In all cases, some way is needed to ensure that adequate exploration occurs while still allowing convergence to an optimal policy. The first method we describe is the generic indirect method that combines system identification and uses conventional DP at each time step. Although this method's computational complexity limits its utility, it serves as a reference point for comparative purposes and gives us the opportunity to describe the system identification method that tends to be used in indirect methods for these problems. We then describe another indirect method that is the simplest modification of the generic method employing real-time DP. We call this method *adaptive real-time DP*. The third method we describe is the direct Q-learning method of Watkins [67]. Finally, we briefly describe hybrid direct/indirect methods.

7.3 The Generic Indirect Method

Indirect methods for adaptive Markovian decision problems explicitly estimate the unknown state-transition probabilities and immediate costs based on the history of state transitions and immediate costs observed while the controller and system interact. The usual approach is to define the state-transition probabilities in terms of a parameter, θ, contained in some parameter space, Θ. Thus, for each pair of states $i, j \in S$ and each action $u \in U(i)$, $p(i, j, u, \theta)$ is the state-transition probability corresponding to parameter $\theta \in \Theta$, where the functional dependence on θ has a known form. Further, one assumes that there is some $\theta^* \in \Theta$ that is the true parameter, so that $p_{ij}(u) = p(i, j, u, \theta^*)$. The identification task is to estimate θ^* from experience. The usual approach takes as the estimate of θ^* at each time step the parameter having the highest probability of generating the observed history, i.e., the maximum-likelihood estimate of θ^*.

The simplest form of this approach to identification is to assume that the unknown parameter is simply a list of the actual transition probabilities. Then at each time step t the system model consists of the maximum-likelihood estimates, denoted $p^t_{ij}(u)$, of the unknown state-transition probabilities of all pairs of states i, j and actions $u \in U(i)$. These estimates are formed by keeping track over time of how frequently the various state transitions occur when the various actions are generated. Specifically, let $n^u_{ij}(t)$ be the observed number of times before time step t that action u was generated when the system was in state i and made a transition to state j. Then $n^u_i(t) = \sum_{j \in S} n^u_{ij}(t)$ is the number of times action u was generated in state i. The estimates at time t of the unknown state-transition probabilities, which constitute the maximum-likelihood system model at time t, are

$$p^t_{ij}(u) = \frac{n^u_{ij}(t)}{n^u_i(t)}. \tag{7}$$

If the immediate costs, $c_i(u)$, are also unknown, they can be determined simply by memorizing them as they are observed.[11] If in an infinite number of time steps each action would be applied infinitely often in each state, then this system model will converge to the true system. Of course, as discussed above, it is nontrivial to ensure that this occurs at the same time the system is being controlled. Further, convergence to the true system obviously depends critically on being able to observe and identify each state unambiguously .

[11]In problems in which the *expected* immediate cost is a function of the current state and action, the maximum-likelihood estimates of an immediate cost is simply the observed average of the immediate cost for that state and action.

At each time step t, the generic indirect method uses some (non real-time) DP algorithm to determine the optimal evaluation function for the latest system model. Let f_t^* denote this optimal evaluation function. Of course, if the model were correct, then f_t^* would equal f^*, but this is generally not the case. A certainty equivalence optimal policy for time step t is any policy that is greedy with respect to f_t^*. Let $\mu_t^* = [\mu_t^*(1), \ldots, \mu_t^*(n)]$ denote any such certainty equivalence optimal policy. Then at time step t, $\mu_t^*(s_t)$ is the certainty equivalence optimal action. Any of the off-line DP algorithms described above can be used to determine f_t^*, including asynchronous DP. Here it makes sense to initialize the DP algorithm at each time step with final estimate of f^* produced by the DP algorithm completed at the previous time step. The small change in the system model from time step t to $t+1$ means that f_t^* and f_{t+1}^* will probably also not differ significantly so that the DP algorithm will tend to converge after few iterations. As pointed out above, however, the computation required to perform even one iteration can be prohibitive in problems with large numbers of states.

What action should the controller generate at time t? The certainty equivalence optimal action, $\mu_t^*(s_t)$, *appears* to be the best based on observations up to time t. Consequently, in pursuing its objective of *control*, the controller should always generate this action. However, because the current model is not necessarily correct, the controller must also pursue the *identification* objective, which dictates that it must sometimes select actions *other* than the certainty equivalence optimal actions. It is easy to generate examples in which always following the current certainty equivalence optimal policy prevents convergence to a true optimal policy due to lack of exploration (see, for example, Kumar [33]).

One of the simplest ways to induce exploratory behavior is to make the controller use randomized policies in which actions are chosen according to probabilities that depend on the current certainty equivalence optimal evaluation function. Each action always has a nonzero probability of being generated, with the current certainty equivalence optimal action having the highest probability. To facilitate comparison of algorithms in the simulations described in Section 11, we adopt the action-selection method based on the Boltzmann distribution that was used by Watkins [67], Lin [39], Singh [54], and Sutton [58]. This method assigns to each admissible action for the current state a probability of its being generated, where this probability is determined by a rating of each action's utility. The Q-value (Equation 2) of the action for the current state and the most recent estimate of f^* provides the appropriate rating of utility. Assuming, as in Section 6.1, that the estimate of f^* has already been updated to f_{t+1}^* by the time action u_t must be generated, we rate each action $u \in U(s_t)$ as follows:

$$r(u) = Q^{f_{t+1}^*}(s_t, u).$$

We then transform these ratings (which can be negative and do not sum to one) into a probability mass function over the admissible actions using the Boltzmann distribution: at time step t, the probability that the controller generates action $u \in U(s_t)$ is

$$\mathrm{Prob}(u) = \frac{e^{-r(u)/T}}{\sum_{v \in U(s_t)} e^{-r(u)/T}}, \tag{8}$$

where T is a positive parameter controlling how sharply these probabilities peak at the certainty equivalence optimal action, $\mu_t^*(s_t)$. As T increases, these probabilities become more uniform, and as T decreases, the probability of generating $\mu_t^*(s_t)$ approaches one, while the probabilities of the other actions approach zero. T acts as a kind of "computational temperature" as used in optimization based on simulated annealing [29] in which T is decreased over time. Here it controls the necessary tradeoff between identification and control. At "zero temperature" there is no exploration, and the randomized policy equals the certainty equivalence optimal policy, whereas at "infinite temperature" there is no attempt at control.

In the simulations described in Section 11, we introduced exploratory behavior by using the method just described for generating randomized policies, and we let T decrease over time as learning progressed. Our choice of this method was dictated by simplicity and our desire to illustrate algorithms that are as "generic" as possible. Without doubt, more sophisticated means of inducing exploratory behavior, as suggested by the authors cited in Section 7.2, would have beneficial effects on the behavior of these algorithms.

7.4 Adaptive Real-Time Dynamic Programming

The generic indirect method just presented relies on executing a non real-time DP algorithm until convergence at each time step. It is straightforward to substitute real-time DP. The result is another indirect method which we call *adaptive real-time DP*. Specifically, this method is exactly the same as real-time DP as described in Section 6.2 except that 1) a system model is updated using some on-line system identification method, such as the method given by Equation 7; 2) the current system model is used in performing the stages of real-time DP instead of the true system model; and 3) the action at each time step is determined by the randomized policy given by Equation 8, or by some other method that balances the identification and control objectives.

Adaptive real-time DP is related to a number of algorithms that have been investigated by others. Although Sutton's *Dyna* architecture [58] focuses on Q-learning and methods based on policy iteration (Section 8), it also encompasses algorithms such as adaptive real-time DP, as he discusses in ref. [59]. Lin [39, 38] also discusses methods closely related to adaptive real-time DP. In the field of control theory, Jalali and Ferguson [24] describe an algorithm that is essentially adaptive real-time DP, although they focus on Markovian decision problems in which performance is measured by the average cost per-time-step instead of the discounted cost we have discussed.

Performing real-time DP concurrently with system identification, as in adaptive real-time DP, provides an opportunity to let progress in identification influence the states whose costs are backed up during each time interval. Sutton [58] suggested that it can be advantageous to back up the costs of states for which there is good confidence in the accuracy of the estimated state-transition probabilities. One can devise various measures of confidence in these estimates and direct the algorithm to the states whose cost backups use the most reliable state-transition information according to this confidence measure. At the same time, it is possible to let this kind of confidence measure direct the selection of actions so that the controller tends to visit regions of the state space where the confidence is *low* so as to improve the model for these regions. This strategy produces exploration that aids identification but can conflict with control as discussed in Section 7.2. Kaelbling [27], Lin [39], Moore [44], Schmidhuber [49], Sutton [58], and Thrun and Möller [65] discuss these and other possibilities.

7.5 Q-Learning

Q-learning is a method for solving adaptive Markovian decision problems proposed by Watkins [67]. Unlike the indirect adaptive methods discussed above, it is a direct method because it does not use an explicit model of the dynamical system underlying the decision problem. Instead it directly estimates the optimal Q-values for all pairs of states and admissible actions. Recall from Equation 3 that $Q^*(i,u)$, the optimal Q-value for state i and action $u \in U(i)$, is the cost of generating action u in state i and thereafter following an optimal policy. Any greedy action with respect to optimal Q-values for a state is an optimal action. Thus, if the optimal Q-values are available, an optimal policy can be determined with little computation. Watkins [67] actually proposed a family of Q-learning methods, and what we call Q-learning in this article is the simplest special case, which he called "one-step Q-Learning." Watkins observed that although Q-learning methods are based on a simple idea, they had not been suggested previously as far as he knew. He further observed, however, that because these problems had been so intensively studied for over thirty years, it would be surprising if no one had considered these methods earlier. We have not yet seen discussions of them that predate his 1989 dissertation.

In our presentation of Q-learning, we depart somewhat from the view taken by Watkins [67] and others (e.g., Sutton [58], Barto and Singh [5]) of Q-learning as a method for adaptive on-line control. In order to take maximal advantage of the theoretical perspective adopted in this article and to emphasize Q-learning's relationship to asynchronous DP, we first present the basic Q-learning algorithm as an *off-line* asynchronous DP method that is unique in not requiring direct access to the state-transition probabilities of the decision problem. We then present the more usual on-line view of Q-learning.

7.5.1 Off-Line Q-Learning

Off-Line Q-learning works as follows. Instead of maintaining an explicit estimate of the optimal evaluation function, as is done by all the methods described above, Q-learning maintains an estimate of the optimal Q-values for each state and admissible action. For any state i and action $u \in U(i)$, let $Q_k(i,u)$ be the estimate of $Q^*(i,u)$ available at stage k of the computation. Recalling that f^* is the minimum of the optimal Q-values for each state (Equation 4), we can think of the Q-values at stage k as implicitly defining f_k, a stage-k approximation of f^*, which is defined for each state i by

$$f_k(i) = \min_{u \in U(i)} Q_k(i,u). \tag{9}$$

Although Q-values define an evaluation function in this way, they contain more information than the evaluation function. Unlike the complete set of Q-values for a state, the evaluation function does not itself contain information about the costs of actions that are second best, third best, etc.,[12] information that can be useful in a variety of situations as illustrated by Sutton [58].

Instead of having direct access to the state-transition probabilities $p_{ij}(u)$, for all states i, j, and all actions $u \in U(i)$, off-line Q-learning has access only to a random function that can generate samples according to these probabilities. Thus, if this function is given a state i and an action $u \in U(i)$, it returns a state j with probability $p_{ij}(u)$. Let us call this function **successor** so that $j = \mathtt{successor}(i,u)$. The **successor** function amounts to an accurate model of the system in the form of its state-transition probabilities, but the algorithm does not have access to the probabilities themselves.[13] As we shall see below, the role of the **successor** function is played by the system itself in on-line Q-learning.

At each stage k, off-line Q-learning synchronously updates the Q-values of a subset of the state-action pairs and leaves unchanged the Q-values for the other state-action pairs. The subset of state-action pairs whose Q-values are updated changes from stage to stage, and the choice of these subsets determines the precise nature of the algorithm. For each $k = 0, 1, \ldots$, let $S_k^Q \subseteq \{(i,u) | i \in S, u \in U(i)\}$ denote the set of state-action pairs whose Q-values are updated at stage k. For each state-action pair in S_k^Q, it is necessary to define a learning rate parameter that determines how much of the new Q-value is determined by its old value and how much by a backed-up value. Let $\alpha_k(i,u)$, $0 < \alpha_k(i,u) < 1$, denote the learning rate parameter for updating the Q-value of (i,u) at stage k. Then Q_{k+1} is computed as follows:

[12]Ranking actions using the evaluation function requires knowledge of the state-transition probabilities.

[13]Watkins [67] defined Q-learning for the case in which the immediate costs are also determined probabilistically from state-action pairs. In this case, the algorithm also requires access to random samples with expected values $c_i(u)$.

if $(i,u) \in S_k^Q$ then

$$Q_{k+1}(i,u) = (1-\alpha_k(i,u))Q_k(i,u) + \alpha_k(i,u)[c_i(u) + \gamma f_k(\texttt{successor}(i,u))]. \tag{10}$$

The Q-values for the other state-action pairs remain the same, i.e.,

$$Q_{k+1}(i,u) = Q_k(i,u),$$

for all $(i,u) \notin S_k^Q$. This sequence of learning rate parameters for each (i,u) has to decrease over the stages in a certain way for the algorithm to converge as described below. When $\alpha_k(i,u) = 1$, the backed-up Q-value simply replaces the old, and as $\alpha_k(i,u)$ decreases, a decreasing fraction of the backed-up value contributes to the update.

One can gain some insight into off-line Q-learning by relating it to asynchronous DP. The stage-k Q-values for all relevant state-action pairs define the evaluation function f_k given by Equation 9. Thus, one can view a stage of off-line Q-learning defined by Equation 10 as updating f_k to f_{k+1}, where for each state i,

$$f_{k+1}(i) = \min_{u \in U(i)} Q_{k+1}(i,u).$$

This evaluation function update does not correspond to a stage of any of the usual DP algorithms because it is based only on samples from `successor` for selected actions determined by the state-action pairs in S_k^Q. Whereas applying the current evaluation function, f_k, to the sample successor state in Equation 10 produces an *unbiased estimate* of the expected successor cost for the *given* admissible action, an asynchronous DP update (Equation 6) uses the *true* expected successor costs over *all* admissible actions. As Watkins [67] pointed out, this is why Q-learning can be viewed as an incremental, Monte Carlo form of DP. We can add that it is a Monte Carlo form of asynchronous DP.

Off-line Q-learning is identical to asynchronous DP in the special case in which 1) the problem is deterministic, and 2) S_k^Q consists of the pairs (i,u) for some fixed i and all $u \in U(i)$, and 3) for all k, $\alpha_k(i,u) = 1$ for all $(i,u) \in S_k^Q$. In this case, stage k of off-line Q-learning uses the costs of the actual successors of state i under all the admissible actions. The result is identical to what would be produced by stage k of asynchronous DP with $S_k = \{i\}$. More generally, in the deterministic case, if S_k^Q consists of all the state-action pairs for states in a set S_k, then stage k of off-line Q-learning has the same effect as the stage of asynchronous DP using S_k. Unfortunately, even with these restrictions on the sets S_k^Q, in the stochastic case an update produced by off-line Q learning is not an unbiased estimate of the corresponding asynchronous DP update.

However, Watkins [67] proved a convergence result for off-line DP. To state this result precisely, it is necessary to place restrictions on the sequences of learning rate parameters for each state-action pair. Because these restrictions must be stated in terms of the number of times the Q-value for each state-action pair is updated, instead of the stage number, we have to appropriately re-index the learning rate parameters. Following Watkins and Dayan [68], let $\alpha_{n_k}(i,u)$ denote the learning rate parameter used for updating the Q-value of (i,u) for the k^{th} time. In other words, n_k is the number of the stage in which the Q-value of (i,u) is updated the k^{th} time. Then the convergence theorem for off-line Q-learning is as follows:

Theorem (Watkins 1989): For $k = 0, 1, \ldots$ and all (i,u), $i \in S$ and $u \in U(i)$, if $0 < \alpha_{n_k}(i,u) < 1$, $\alpha_{n_k}(i,u) \to 0$ as $k \to \infty$, and

$$\sum_{k=0}^{\infty} \alpha_{n_k}(i,u) = \infty, \quad \sum_{k=0}^{\infty} [\alpha_{n_k}(i,u)]^2 < \infty,$$

then the sequence $\{Q_k(i,u)\}$ generated by the off-line Q-learning algorithm converges with probability 1 to $Q^*(i,u)$ as $k \to \infty$.

The proof of this theorem is essentially contained in Watkins' dissertation [67], and Watkins and Dayan present a revised and simplified proof in ref. [68]. The conditions on the sequences of learning rate parameters are standard conditions for related stochastic approximation algorithms (see, e.g., Kasyap, Blaydon, and Fu [28]), and their are a number of simple ways to satisfy them. In Appendix D we describe one method developed by Darken and Moody [13] which we used in obtaining the results for real-time Q-learning on our example problem presented in Section 11. Notice that like the convergence conditions for asynchronous DP, these conditions require the Q-value for each state-action pair to be updated infinitely often in an infinite number of stages

It is not misleading to think of off-line Q-learning as a *more asynchronous version of asynchronous DP*. Asynchronous DP is asynchronous at the level of states, and the backup operation for each selected state requires performing a minimization over all admissible actions for that state. Off-line Q-learning, on the other hand, is asynchronous at the level of state-action pairs.[14] For each selected state-action pair, the system behavior for the other admissible actions for the given state is not used in updating the Q-value. Although it is still necessary to maintain the minimum of the Q-values for each sample successor state over all the admissible actions in order to calculate (via Equation 9) $f_k(\texttt{successor}(i,u))$ used in Equation 10, this can be done without explicit minimization over all the admissible actions at each stage. Because off-line Q-learning explicitly stores Q-values for state-action pairs, it is possible to maintain the required minima incrementally in a way that can take many fewer operations than otherwise required.[15] This advantage is offset by the increased

[14]Williams and Baird [79] discuss DP algorithms that are asynchronous at an even finer grain than is Q-learning.

[15]This can be done as follows. Whenever a $Q_k(i,u)$ is updated, if its new value, $Q_{k+1}(i,u)$, is smaller than $f_k(i)$, then $f_{k+1}(i)$ is set to this smaller value. If its new value is larger than $f_k(i)$, then if $f_k(i) = Q_k(i,u)$

space complexity of Q-learning and the fact that backing up a state's cost in asynchronous DP and updating the Q-value of a state-action pair are not equivalent operations: One would generally expect that many of the latter operations are required to make progress equivalent to the progress made by one of the former. Nevertheless, this property of off-line Q-learning can be advantageous when backups have to be accomplished quickly—as in real-time applications—despite a large number of admissible actions.

7.5.2 Real-Time Q-Learning

One can perform off-line Q-learning on-line by interleaving its stages with control steps in much the same way that interleaving the stages of asynchronous DP with control yields real-time DP as discussed in Section 6. If a current system model provides an approximate **successor** function, the result is an indirect adaptive method identical to adaptive real-time DP Section 7.4 except that stages of off-line Q-learning substitute for stages of asynchronous DP. As mentioned above, this method might have certain kinds of advantages over adaptive real-time DP when there are a large number of admissible actions. However, we use the term *real-time Q-learning* for the case originally discussed by Watkins [67] in which there is no model of the system underlying the decision problem and the real system plays the role of the **successor** function. This direct adaptive algorithm updates the Q-value for only a single state-action pair at each time step of control, where this state-action pair consists of the observed current state and the action actually generated.

Specifically, assume that at each time step t the controller observes state s_t and has available the estimated optimal Q-values produced by all the preceding stages of on-line Q-learning. Denote these estimates $Q_t(i,u)$ for states i and admissible actions u. Using this information in some manner that allows for exploration, the controller generates an action $u_t \in U(s_t)$, which is input to the system. The controller receives the immediate cost $c_{s_t}(u_t)$ while the system state changes to s_{t+1}. Then Q_{t+1} is computed as follows:

$$Q_{t+1}(s_t,u_t) = (1-\alpha_t(s_t,u_t))Q_t(s_t,u_t) + \alpha_t(s_t,u_t)[c_{s_t}(u_t) + \gamma f_t(s_{t+1})], \qquad (11)$$

where $f_t(s_{t+1}) = \min_{u \in U(s_{t+1})} Q_t(s_{t+1},u)$ and $\alpha_t(s_t,u_t)$ is the learning rate parameter at time step t for the current state-action pair. The Q-values for all the other state-action pairs remain the same, i.e,

$$Q_{t+1}(i,u) = Q_t(i,u),$$

and $f_k(i) \neq Q_k(i,u')$ for any $u' \neq u$, then $f_{k+1}(i)$ is found by explicitly minimizing the current Q-values for state i over the admissible actions. This is the case in which u is the sole greedy action with respect to $f_k(i)$. Otherwise, nothing is done, i.e., $f_{k+1}(i) = f_k(i)$. This procedure therefore computes the minimization in Equation 9 explicitly only when updating the Q-values for state-action pairs (i,u) in which u is the sole greedy action for i and the Q-value increases.

for all $(i, u) \neq (s_t, u_t)$. This process repeats for each time step.

Real-time Q-learning is the special case of off-line Q-learning in which S_t^Q, the set of state-action pairs whose Q-values are updated at each step (or stage) t, is $\{(s_t, u_t)\}$. Thus, the sequence of Q-values generated by real-time Q-learning converges to the true values given by Q^* if the sequences of learning rate parameters satisfy the conditions required by the off-line Q-learning theorem. An implication of these conditions is that in an infinite number of control steps each admissible action must be performed in each state infinitely often. This last condition is the same as required for convergence of the maximum likelihood model used in the indirect adaptive methods described in Section 7.3. It is also noteworthy as pointed out by Dayan [15], that when there is only one admissible action for each state, real-time Q-learning reduces to the TD(0) algorithm investigated by Sutton [57].

To define a complete adaptive control algorithm making use of real-time Q-learning it is necessary to specify how each action is generated based on the current Q-values. Convergence to an optimal policy requires the same kind of exploration required by indirect methods to facilitate system identification as discussed in Section 7.2. Therefore, given a method for generating an action from a current evaluation function, such as the randomized method described above (Equation 8), if this method leads to convergence of an indirect method, it also leads to convergence of the corresponding direct method based real-time Q-learning.

7.5.3 Other Q-Learning Methods

In real-time Q-learning, only the real system underlying the decision problem plays the role of the **successor** function. However, it is also possible to define the **successor** function sometimes by the real system and sometimes by a system model. For state-action pairs actually experienced during control, the real system provides the **successor** function; for other state-action pairs, a system model provides an approximate **successor** function. Sutton [58] has studied this approach in an algorithm called *Dyna-Q*, which performs the basic Q-learning update using both actual state transitions as well as hypothetical state transitions simulated by a system model. Using the Q-learning update on hypothetical state transitions amounts to running multiple stages of off-line Q-learning in the intervals between times at which the controller generates actions. A step of real-time Q-learning is performed based on each actual state transition. This is obviously only one of many possible ways to combine direct and indirect adaptive methods as emphasized in Sutton's discussion of the general Dyna learning architecture [58]. For example, another method interleaves stages of adaptive real-time DP with the steps of real-time Q-learning.

It is also possible to modify the basic Q-learning method in a variety of ways in order to enhance its efficiency. For example, Lin [39] has studied a method in which real-time Q-learning is augmented with model-based off-line Q-learning only if one action does not clearly stand out as preferable according to the current Q-values. In this case, off-line Q-learning is carried out to update the Q-values for all of the admissible actions that are "promising" according to the latest Q-values for the current state. Watkins [67] describes a family of Q-learning methods in which Q-values are updated based on information gained over sequences of state transitions (although his convergence proof does not apply to these more general algorithms). One way to implement this kind of extension is to use the "eligibility trace" idea (refs. [2, 31, 56, 60, 57]) to update the Q-values of all the state-action pairs experienced in the past, with the magnitudes of the updates decreasing to zero with increasing time in the past. Sutton's [57] TD(λ) class of algorithms illustrate this idea. Attempting to present all of the combinations and variations of Q-learning methods that have been, or could be, described is well beyond the scope of the present article. Barto and Singh [5], Dayan [14, 15], Lin [39, 38], and Sutton [58] present comparative empirical studies of some of the adaptive algorithms based on Q-learning.

8 Methods based on Explicit Policy Representations

All of the DP-based learning algorithms described above, for both non-adaptive and adaptive problems, use an explicit representation of either an evaluation function or a function giving the Q-values of state-action pairs. These functions are used in computing the action at each time step, but the policy so defined is not explicitly stored. There are a number of other real-time learning and control methods based on DP in which policies as well as evaluation functions are stored and updated at each time step of control. Unlike the methods addressed in this article, these methods are based on the *policy iteration* DP algorithm rather than on the value iteration algorithm discussed in Section 5. Policy iteration (see, e.g., Bertsekas [8]) alternates two basic stages in which 1) the evaluation function for the current policy is determined (either by a successive approximation method similar to value iteration but not requiring its minimization step, or by matrix inversion), and 2) the current policy is updated to be a greedy policy with respect to the current evaluation function. Real-time algorithms based on policy iteration work by interleaving the computations involved in these stages with each other and with the generation of control actions. They interleave stages in which 1) the current evaluation function is updated to better approximate the evaluation function for the current policy, 2) the current policy is updated to select actions that are better according to the current evaluation function, and 3) an action for the current state is generated based on the current policy. Unlike the usual policy iteration algorithm, real-time algorithms based on policy iteration do not completely determine the evaluation function for each new policy before they update that policy.

Examples of such methods appear in the pole-balancing system of Barto, Sutton, and Anderson [2, 56] (also refs. [1, 56]) and the *Dyna-PI* method of Sutton [58] (where PI means Policy Iteration). Barto, Sutton, and Watkins [4, 3] discuss the connection between these methods and policy iteration in detail. In this article we do not discuss real-time algorithms based on policy iteration because their theory is not yet as well understood as is the theory of real-time algorithms based on asynchronous value iteration. However, Williams and Baird [79] have made a valuable contribution to this theory by addressing DP algorithms that are asynchronous at a finer grain than asynchronous DP and Q-learning. These algorithms include both value iteration and policy iteration as special cases. Integrating their theory with that presented here is beyond the scope of this article.

9 Storing Evaluation Functions

An issue of great practical importance in implementing any of the algorithms described in this article is how evaluation functions are represented and stored.[16] All of the theoretical results we have described assume that the cost of each state is *explicitly* stored. We refer to this as the ***lookup-table representation***, which—at least in principle—is always possible when there are a finite number of states and actions, as assumed throughout this article. In applying conventional DP to problems involving continuous states and/or actions, the usual practice is to discretize the ranges of the continuous state variables and then use the lookup-table representation (cf. the "boxes" representation used by Michie and Chambers [41] and Barto, Sutton, and Anderson [2]). This leads to space complexity exponential in the number of state variables, the situation prompting Bellman [6] to coin the phrase "curse of dimensionality." The methods described in this article based on asynchronous DP and Q-learning do not escape this problem.

A number of methods exist for making the lookup-table representation more efficient when it is not necessary to store the costs of all possible states. Hash table methods, as assumed by Korf [32] for LRTA*, permit efficient storage and retrieval when the costs of a small subset of the possible states need to be stored. Similarly, using the ***kd-tree*** data structure to store state costs, as explored by Moore [44, 45], can provide efficient storage and retrieval of the costs of a finite set of states from a k-dimensional state space. The theoretical results described in this article extend to this method because it preserves the integrity of each stored cost. These results also extend to the hash table method under the condition that no hash collisions occur. Methods such as these are particularly appropriate for problems, such as the example problem described in the next section, in which real-time algorithms focus on increasingly small subsets of states. Stored cost estimates for states rarely visited after the early stages of learning can be over-written by cost estimates of states on which the algorithm focuses in later stages.

Other approaches to storing evaluation functions use parametric function approximation methods. For example, in Samuel's [47] application of a method similar to real-time DP to

[16]All of our comments here also apply to storing the Q-values of admissible state-action pairs.

the game of checkers, the evaluation function was approximated as a weighted sum of the values of a set of features describing checkerboard configurations. The basic backup operation was performed on the weights, not on the state costs themselves. The parameters specifying the approximate evaluation function, that is, the weights, were adjusted to reduce to the discrepancy between the current cost of a state and its backed-up cost obtained by applying the current evaluation function to the successor states. This approach inspired a variety of more recent studies using parameterized function approximations. The discrepancy supplies the error for any error-correction procedure that approximates functions based on a training set of function samples. This is a form of supervised learning, or learning from examples, and provides the natural way to make use of connectionist networks as shown, for example, by Anderson [1]. Parametric approximations of evaluation functions are useful because they can generalize beyond the training data to supply cost estimates for states that have never before been visited, an important factor for large state sets. Combining DP and parametric function approximation is discussed by Barto, Sutton, and Watkins [4] and Watkins [67].

In fact, almost any supervised learning method, and its associated manner of representing hypotheses, can be adapted for approximating evaluation functions. This includes symbolic methods for learning from examples. These methods also generalize beyond the training information, which is derived from the back-up operations of the various DP-based algorithms we have described. For example, Chapman and Kaelbling [12] and Tan [64] adapt decision-tree methods for learning evaluation functions.

Despite the large number of studies in which the principles of DP have been combined with generalizing methods for approximating evaluation functions, *the theoretical results presented in this article do not automatically extend to these approaches.* Although generalization can be helpful in approximating the optimal evaluation function, it is often detrimental to the convergence of the underlying asynchronous DP algorithm, as pointed out by Watkins [67]. Even if a function approximation scheme can adequately represent the optimal evaluation function when trained on samples from this function, it does not follow that an adequate representation will result from an iterative DP method that uses such an approximation scheme at each stage. The issues are much the same as those that arise in the numerical solution of differential equations.[17] The objective of these problems is to approximate the function that is the solution of a differential equation for given boundary conditions in the absence of training examples drawn from the unknown true solution. In other words, the objective is to *solve approximately* the differential equation, not just to approximate its solution. Here, we are interested in approximately solving the Bellman Optimality Equation and not merely in approximating a given solution.

There is an extensive literature on numerical approximation methods applied to optimal

[17]Indeed, in the case of continuous time and continuous state space, the optimal evaluation function is the solution of a partial differential equation (known as the Hamilton-Jacobi-Bellman Equation) which is the counterpart of the Bellman Optimality Equation (Equation 4).

control, such as finite element methods and methods using orthogonal polynomials (e.g., Boudarel et al. [10] and Gonzalez and Rofman [18]). However, most of this literature is devoted to off-line algorithms for approximating optimal evaluation functions defined on infinite state spaces when there is a complete model of the decision problem. Borrowing techniques from this literature to produce approximation methods for real-time DP in both adaptive and non-adaptive problems is a challenge for future research.

To the best of our knowledge, the only theoretical results directly relevant to using generalizing methods with adaptive real-time algorithms based on DP are those of Sutton [57], which apply when states are represented by a linearly independent set of vectors. These results concern the trial-based solution to the problem of using a TD method to approximate the evaluation function of a fixed policy as a linear function of the vectors representing the states. Dayan [15] generalized this result to a more general class of TD methods. Much more research is needed to provide a better understanding of how function approximation methods can be used effectively with the algorithms described in this article.

10 The Race Track Problem

To illustrate the algorithms discussed above we applied them to a game described by Martin Gardner [16] called Race Track, which simulates automobile racing. We chose this game because it is an interesting optimal control problem that is easy to describe and has finite state and action sets. We modified the game as described by Gardner by considering only a single car, making it probabilistic, bounding the velocities, and handling collisions with the track boundaries differently.

A race track of any length and shape is drawn on graph paper, with a starting line at one end and a finish line at the other consisting of designated squares. Each square within the boundary of the track is a possible location of the car. At the start of each trial, the car is placed on the starting line, and moves are made in which the car attempts to move down the track toward the finish line. Acceleration and deceleration are simulated as follows. If in the previous move the car moved h squares horizontally and v squares vertically, then the present move can be h' squares vertically and v' squares horizontally, where the difference between h' and h is -1, 0, or 1, and the difference between v' and v is -1, 0, or 1. This means that the car can maintain its speed in either direction, or it can slow down or speed up in either direction by one square per move. However, to make the problem have a finite number of states, we imposed a speed limit L on the car: its speed in any direction cannot exceed L. To impose this limit, we modified the above acceleration rule so that if the magnitude of h' (or v') as computed above exceeds L, then h' (or v') is changed so that its magnitude is L (and its sign is unchanged). The objective is to cross the finish line in as few moves as possible.

We made the problem probabilistic by making the effect of the car's acceleration depend on a random factor. At each move, the intended acceleration in either the horizontal or vertical directions can be -1, 0, or 1. With a probability p, the actual accelerations at each move are zero independently of the intended accelerations. Thus, $1-p$ is the probability that the controller's intended actions are executed. One might think of this as simulating driving on a track that is unpredictably slippery so that sometimes braking and throttling up have no effect on the car's velocity.

Collisions with the track boundaries are handled as follows. If the projected path of the car for a move is determined to intersect the edge of the race track at any place not on the finish line, the car moves to that intersection point, its speed is reduced to zero in each direction (i.e., $h'-h$ and $v'-v$ are considered to be zero), and the trial continues. This implies that the car never leaves the track except by crossing the finish line and that it should usually avoid collisions with the edge of the track to achieve fast finish times (although in some cases optimal policies exist that cause collisions).

The first step in formulating this game as an example of a stochastic optimal path problem is to define the dynamical system being controlled. The state of the car at each time step $t = 0, 1, \ldots$ is a quadruple of integers $s_t = (x_t, y_t, \dot{x}_t, \dot{y}_t)$. The first two integers are the horizontal and vertical coordinates of the car's location, and the second two integers are its speeds in the horizontal and vertical directions. That is, $\dot{x}_t = x_t - x_{t-1}$ is the horizontal speed of the car at time step t; similarly $\dot{y}_t = y_t - y_{t-1}$ (we assume $x_{-1} = y_{-1} = 0$). Because there are a finite number of possible locations for the car and the speed limit is strictly enforced, the set of possible states is finite. The set of admissible actions for each state is the set of pairs (u^x, u^y), where u^x and u^y are both in the set $\{-1, 0, 1\}$. We let $u_t = (u_t^x, u_t^y)$ denote the action at time t.

The following equations define the state transitions of this system. With probability p, the state at time step $t+1$ is

$$\begin{aligned} x_{t+1} &= x_t + \dot{x}_t \\ y_{t+1} &= y_t + \dot{y}_t \\ \dot{x}_{t+1} &= \dot{x}_t \\ \dot{y}_{t+1} &= \dot{y}_t, \end{aligned} \tag{12}$$

and with probability $1-p$, the state at time step $t+1$ is

$$\begin{aligned} x_{t+1} &= x_t + \dot{x}_t + u_t^x \\ y_{t+1} &= y_t + \dot{y}_t + u_t^y \\ \dot{x}_{t+1} &= \mathrm{chop}_L(\dot{x}_t + u_t^x) \\ \dot{y}_{t+1} &= \mathrm{chop}_L(\dot{y}_t + u_t^y), \end{aligned} \tag{13}$$

where

$$\mathrm{chop}_L(z) = \begin{cases} -L & \text{if } z < -L \\ L & \text{if } z > L \\ z & \text{otherwise,} \end{cases}$$

imposes the speed limit of $L > 0$.

This assumes that the straight line joining the point (x_t, y_t) to the point (x_{t+1}, y_{t+1}) lies entirely within the track, or intersects only the finish line. If this is not the case, then let (c_t^x, c_t^y) be the (first) point of intersection between the track boundary and line joining the (x_t, y_t) and (x_{t+1}, y_{t+1}), where the latter coordinates are given by Equations 12 or 13. In this case, the state at time $t+1$ is simply set to $(c_t^x, c_t^y, 0, 0)$ with probability one, i.e., the car always comes to a stop exactly at the intersection. A move that takes the car across the finish line is treated as a valid move, but we assume that the car subsequently stays in the resulting state until a new trial begins. This method for keeping the car on the track, together with Equations 12 and 13, define the state-transition probabilities for all states and admissible actions.

To complete the formulation of a stochastic optimal path problem, we need to define the set of start states, the set of goal states, and the immediate costs associated with each action in each state. The set of start states consists of all the zero-velocity states on the starting line, i.e., all the states $(x, y, 0, 0)$ where (x, y) are coordinates of the squares making up the starting line. The set of goal states consists of all states that can be reached in one time step by crossing the finish line from inside the track. According to the state-transition function defined above, this set is absorbing. The immediate cost for all non-goal states is 1.0 independently of the action taken, i.e., $c_i(u) = 1.0$ for all non-goal states i and all admissible actions u. The immediate cost associated with a transition from any goal state is 0.0. According to these immediate costs, a policy that minimizes the expected total infinite-horizon undiscounted cost is a policy by which the car crosses the finish line as quickly as possible starting from any state.

11 Simulation Results

To illustrate and compare their performances, we applied four DP-based algorithms to an example of the race track problem. Gauss-Seidel DP and real-time DP apply to the non-adaptive case; adaptive real-time DP and real-time Q-learning apply to the adaptive case. Although real-time DP and adaptive real-time DP can back up the costs of many states at each control step, we restricted attention to the simplest case in which they only back up the cost of the current state at each time step. This is the case in which $B_t = \{s_t\}$ for all t.

The example race track is shown in Figure 2. With the speed limit, L, set at 6, there are 8811 states, four of which are start states and 87 of which are goal states. We set $p = 0.1$ so that the controller's intended actions were executed with probability 0.9. We set the problem up to satisfy the hypotheses of the Trial-Based Real-Time DP Theorem given in Section 6.2. It is clear that there is at least one proper policy: it is possible for the car to reach the finish line from any initial state. Because all the immediate costs are positive, we know that $f^*(i)$ must be non-negative for all states i. Thus, setting the initial costs of all the states to zero produces a non-overestimating initial evaluation function as required by the theorem. We applied the real-time algorithms in a trial-based manner, starting each trial with the car placed on the starting line with zero velocity, where each square on the starting line was selected with equal probability, and ending each when the car reached a goal state. Thus, real-time DP with the discount factor γ equal to one will converge to the optimal evaluation function with repeated trials. Gauss-Seidel DP executed off-line with $\gamma = 1$ also converges under these conditions.

Of the four algorithms, only Gauss-Seidel DP, by which we mean Gauss-Seidel value iteration as defined in Section 5.2, is a deterministic off-line algorithm. Because the real-time algorithms are stochastic, we ran each of them several times with different random number seeds, and the results we report are typical of these runs. To monitor the improvement in control performance produced by the three real-time algorithms, we interspersed *testing trials* with *training trails*. Each training trial began and ended as described above, and the algorithm was executed during each training trial to update the evaluation function. Each testing trial was run with the evaluation function fixed at the evaluation function resulting from the preceding training trial, i.e., learning was turned off, and the car followed the greedy policy with respect to the current evaluation function, i.e., the random exploration used by the adaptive algorithms was turned off. Like the training trials, each testing trial began with the car placed at a randomly chosen state in the set of start states, but a time-out mechanism was needed because with the evaluation function fixed and exploration turned off the car could become trapped in a loop. A testing trial ended when the car reached a goal state or when 500 moves had been taken since the beginning of the trial. By an *epoch* we mean a sequence of trials consisting of 20 training trials followed by 500 testing trials. The parameter settings and other details of the simulations are given in Appendix D.

Figure 2 shows results from a typical run of real-time DP (Panel A), adaptive real-time DP (Panel B), and real-time Q-learning (Panel C). The graph in each panel shows the path length, i.e., the number of moves required to reach the finish time, averaged over the testing trials of each epoch as a function of the number of epochs for the corresponding algorithm. Note that the tops of the graphs in Panels B and C have been clipped at the scale chosen. Many of the paths generated in the testing trials of the early epochs of these runs required considerably more than 200 moves, and many timed out at the 500 move limit. It is clear from the graphs that in this problem, real-time DP learns faster than adaptive real-time DP, which learns faster than real-time Q-learning, when learning rate is measured in terms of the number actions executed. This is not surprising given the nature of the algorithms and the differences between the non-adaptive (Panel A) and the adaptive (Panels B and C) versions of the problem.

The right side of each panel of Figure 2 shows the paths followed by the car from each start state after a number of epochs that differed for each algorithm. These paths were generated by *always* applying the actions generated by the controller, instead of applying them with probability $1 - p$. The paths resulted from 150 epochs (Panel A), 400 epochs (Panel B), and 1,000 epochs (Panel C) of learning for the corresponding algorithms. The paths shown in Panels A and B are examples of optimal paths, while it is clear that the paths shown in Panel C are not. Real-time Q-learning took about 2,500 epochs to produce optimal paths.

Table 1 summarizes the results. Gauss-Seidel DP was considered to have converged to the optimal evaluation function after 20 sweeps, which required a total of 174,480 backups. At this point, the maximum cost change over all states between two successive sweeps was less than 10^{-4}. However, we also determined that the evaluation function produced after 11 sweeps, or 95,964 backups, was good enough so that the corresponding greedy policy was an optimal policy. We did this by running testing trials after each sweep, much as we did in testing the real-time algorithms. The optimal trajectory produced by this optimal policy, which still varies due to the underlying stochasticity in the problem, takes an average of 12.23 moves. It is important to note, however, that this repeated testing is not a part of the basic Gauss-Seidel DP algorithm, or of any off-line value iteration algorithm. Without interleaved testing, there is no way to estimate when the evaluation function specifies an optimal policy before it converges to the optimal evaluation function.[18]

Of the other algorithms, real-time DP is most directly comparable to Gauss-Seidel DP. After about 150 epochs, or 3,000 training trials, real-time DP improved control performance to the point where a trial took an average of 12.6 moves. After this number of epochs, performance continued to improve, but much more slowly (see Figure 2, Panel A). We therefore considered the real-time algorithms to have converged when this level of performance was reached, and we use the number of backups required to reach this level of performance as a basis for comparing the real-time algorithms. Real-time DP performed 80,666 backups in reaching this level of performance, about half the number required by Gauss-Seidel DP to converge to the optimal evaluation function. This number of backups also represents some savings over the 95,964 backups in the 11 sweeps of Gauss-Seidel DP after which the resulting evaluation function defines an optimal policy (although the policy formed by real-time DP is not quite optimal at this point).

[18]Policy iteration algorithms address this problem by explicitly generating a sequence of improving policies, but updating a policy requires computing its corresponding evaluation function, which is generally a time-consuming computation. As mentioned in Section 8, real-time algorithms based on policy iteration are beyond the scope of this article.

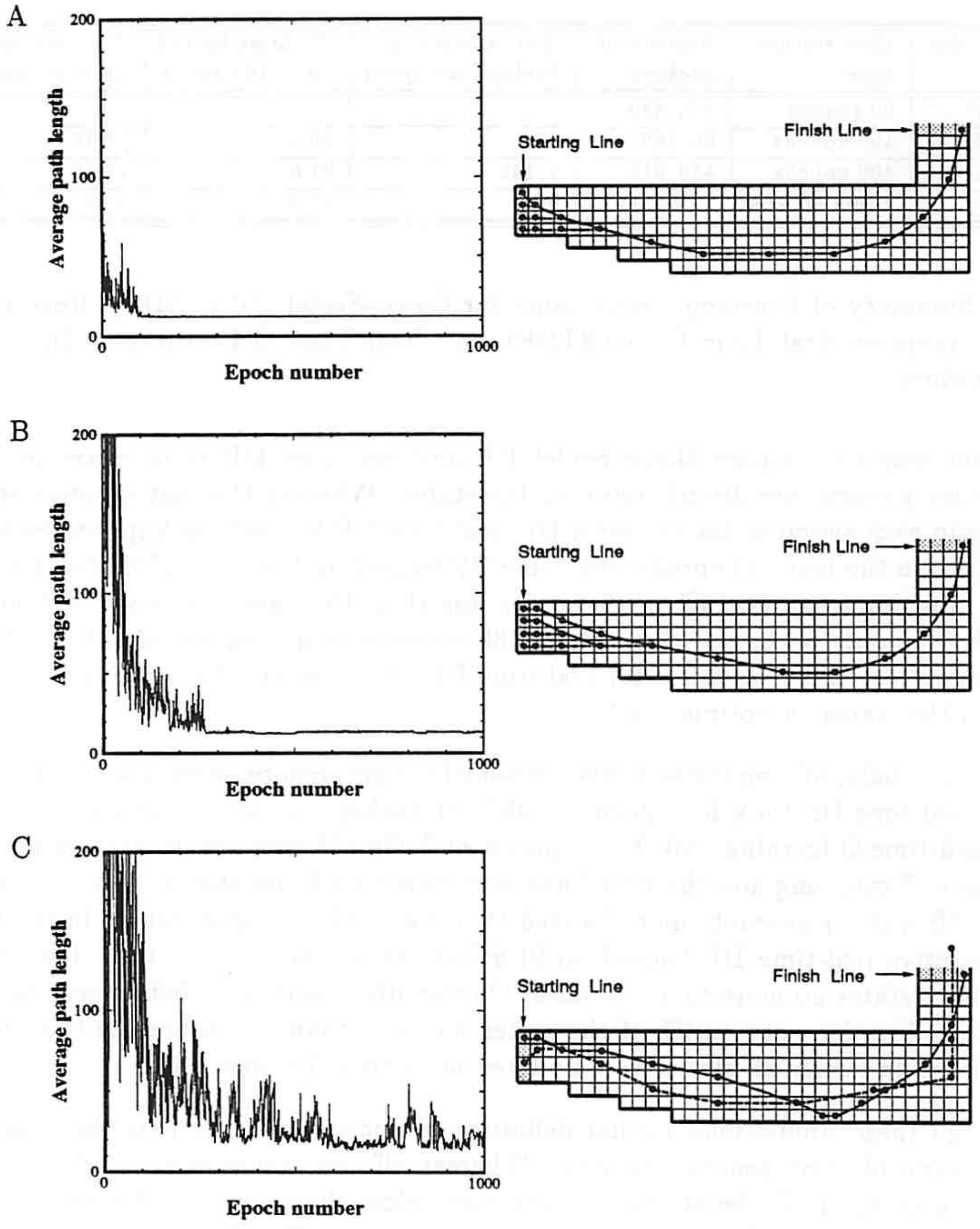

Figure 2: Performance of Three Real-Time Learning Algorithms. Panel A: Real-Time DP. Panel B: Adaptive Real-Time DP. Panel C: Real-Time Q-Learning. Each graph shows the path length averaged over the testing trials of each epoch as a function of the number of epochs for the corresponding algorithm. The right side of each panel shows the race track and the paths followed by the car from each start state after a number of epochs that differed for each algorithm. These paths were generated with random exploration turned off.

Algorithm	*Convergence time*	*Number of backups*	*Ave. number of backups per epoch*	*% states backed up < 100 times*	*% states backed up < 10 times*
GSDP	20 sweeps	174,480	-	-	-
RTDP	150 epochs	80,666	537	99.3	86.15
ARTDP	400 epochs	456,915	1,142	91.6	11.98
RTQ	2,500 epochs	2,356,433	943	40.4	0.02

Table 1: Summary of Learning Performance for Gauss-Seidel DP (GSDP), Real-Time DP (RTDP), Adaptive Real-Time DP (ARTDP), and Real-Time Q-Learning (RTQ). See text for explanation.

Another way to compare Gauss-Seidel DP and real-time DP is to examine how the backups they perform are distributed over the states. Whereas the cost of every state was backed up in each sweep of Gauss-Seidel DP, real-time DP focused backups on fewer states. For example, in the first 150 epochs, real-time DP backed up the costs of 99.3% of the states less than 100 times and 86.15% of the states less than 10 times; the costs of 1,004 states were backed up only once, and the costs of 185 states were not backed up at all. Although we did not collect these statistics for real-time DP after 150 epochs, it became even more focused on the states on optimal paths.

Not surprisingly, solving the adaptive version of the problem requires many more backups. Adaptive real-time DP took 400 epochs, or 456,951 backups, to achieve trials averaging 12.6 moves. Real-time Q-learning took 2,500 epochs, or 2,356,433 backups, to achieve this level of performance. Examining how these backups were distributed over states shows that adaptive real-time DP was considerably more focused than was real-time Q-learning. In the first 400 epochs adaptive real-time DP backed up 91.6% of the states no more than 100 times and 11.98% of the states no more than 10 times. On the other hand, in 2,500 epochs Q-learning updated Q-values for only 40.4% of the states no more than 100 times, and the Q-values corresponding to nearly all states were updated more than 10 times each.

Although these simulations are not definitive comparisons of the four algorithms, they illustrate some of their general features. Whereas off-line Gauss-Seidel DP continued to back up the costs of all the states, the real-time algorithms strongly focused on subsets of the states that were relevant to the control objectives. This focus became increasingly narrow as learning continued. Because the convergence theorem for trial-based real-time DP applies to the simulations of real-time DP, we know that this algorithm eventually would have focused only on states in optimal paths. The adaptive algorithms also focused on progressively fewer states, although more slowly than did real-time DP. Although we did not do so here, one can devise storage methods for state costs or Q-values that exploit the tendency of these algorithms to focus on small subsets of states. Hash table methods would be particularly appropriate as suggested by Korf [32] for LRTA*. Finally, we point out that the amount of computation required by each of the real-time algorithms at each time step was small enough not to have been a limiting factor in the simulations. This is in strong contrast to the amount of computation that would have been required by the generic indirect method, which would have performed an extensive DP iteration at each time step. For this reason, we did not attempt to apply the generic indirect method to this problem.

12 Discussion

An early influence on the field of AI was the observation that algorithms guaranteed to find optimal paths in state-space search require too much time and/or space to be useful for solving important classes of problems. These problems are characterized by very large numbers of states and not enough mathematical structure to permit analytical shortcuts. Abandoning guaranteed optimality, heuristic search algorithms need not expand all possible states because they explore selected solution paths. DP algorithms, developed mainly in control theory and operations research, also avoid exhaustive enumeration of all possible solution paths, but they still require full expansion of all possible states and the storage of a separate cost for each state. This limits their utility for the majority of the problems of interest to AI researchers. Nevertheless, because DP algorithms successively approximate optimal evaluation functions, and explicitly represent each approximation, they are relevant to learning. They effectively cache in a permanent data structure the results of repeated one-step searches forward from each state. This information improves as the algorithm proceeds, converging to the optimal evaluation function, from which one can determine optimal policies with relative ease: decisions that are greedy with respect to the optimal evaluation function are optimal decisions. The result is a closed-loop control policy that specifies an optimal action for each state, i.e., a universal plan based on the states of the underlying dynamical system.

Although the principles of DP are relevant to learning, traditional DP algorithms are not themselves really learning algorithms because they operate off-line. They are not applied *during* problem solving or control, whereas learning occurs as experience accumulates on-line from actual attempts at problem solving or control. However, it is possible to interleave the stages of an otherwise off-line DP algorithm with the steps of on-line problem solving or control, where the stages can be influenced by the observed behavior of the system. We call the resulting algorithm real-time DP. A special case of real-time DP coincides with Korf's LRTA* [32], and this general approach coincides with previous research by others in which DP principles have been used for planning and learning (e.g., refs. [47, 58, 59, 67, 72, 73]).

Our contribution has been to bring to bear on real-time DP the theory of asynchronous DP as presented by Bertsekas and Tsitsiklis [9] and to elaborate Korf's theory of LRTA* within this more general framework. Although the suitability of asynchronous DP for implementation on multi-processor systems motivated the theory described by Bertsekas and Tsitsiklis, we have made use of these results in a manner they did not discuss. Applying these results to real-time DP, especially the results about undiscounted stochastic optimal path problems, provides a more sound theoretical basis for DP-based learning algorithms. Convergence results for asynchronous DP imply that real-time DP retains the competence of conventional synchronous or Gauss-Seidel DP algorithms. Furthermore, when extended using this theory, Korf's LRTA* convergence theorem provides conditions under which real-time DP avoids the exhaustive nature of off-line DP algorithms while still yielding optimal behavior.

We used the formalism of Markovian decision problems to describe algorithms and convergence results. This is the simplest formalism that includes stochastic versions of many of the problems of interest in AI. Stochastic formulations are important due to the uncertainty present in real applications and the fact that uncertainty is what gives closed-loop, or reactive, control advantages over open-loop control. We described two variants of Markovian decision tasks. The first, the non-adaptive case, occurs when there is a complete and accurate model of the decision problem. This model forms the basis for planning, or to use the control theory term, for the design of control policies. In the second kind of Markovian decision task—the adaptive case—this model is lacking.

In non-adaptive problems the advantages of real-time DP arise from several factors. Finding an optimal policy using an off-line DP algorithm may be impractical due to the number of states and actions involved. For example, the off-line DP algorithm known as value iteration successively approximates the optimal evaluation function by repeatedly backing up the costs of all the states until the resulting changes are small enough. Although the estimate of the optimal evaluation function produced at some stage *before value iteration converges* may determine a good, or even an optimal, policy, the algorithm provides no way of knowing when this occurs. One would have to repeatedly interrupt the algorithm during its progress, evaluate the policy specified by the current evaluation function, and decide whether or not this policy is good enough to warrant using it for control. Policy iteration, another off-line DP algorithm (which we did not discuss in detail), explicitly produces a sequence of policies that often converges to an optimal policy before the optimal evaluation function is found. But policy iteration also is not practical for large problems because it requires computing the evaluation function for each current policy, which is a costly computation requiring finding the solution to a set of n simultaneous linear equations, where n is the number of states.

Real-time DP is the result of executing an off-line DP algorithm concurrently with the process of control, where the controller uses the most recent estimate of the optimal evaluation function in deciding on each action. This makes sense when the cost of not acting is higher than the cost of acting suboptimally. As the evaluation function improves over time, the controller automatically makes use of this improving control information. Although each new action taken in a state is not guaranteed to be better than the previous action taken in that state, the overall policy will converge to an optimal policy under the conditions we have given.

Further, the convergence results for asynchronous DP imply that the states whose costs are backed up by real-time DP can be chosen freely to facilitate the gathering of control information. In particular, the choice of these states can be responsive to the current demands of control. A consequence of this is that computational resources can focus on states for which control information is likely to be most important for control performance. Real-time DP is compatible with any exploration scheme designed to facilitate meeting control objectives, such as finding goal states in stochastic optimal path problems. The convergence theorem for trial-based real-time DP as applied to stochastic optimal path problems, which extends Korf's [32] convergence theorem for LRTA* to Markovian decision problems, specifies conditions under which real-time DP focuses on states that are on optimal paths—eventually abandoning all the other states—to produce an *optimal partial policy* without continuing to back up the costs of all the states, and possibly without backing up the costs of some states even once.

Real-time DP is a learning algorithm despite the fact that it requires an accurate model of the decision problem, as emphasized by Korf's [32] choice of the name *Learning* RTA*. Real-time DP accumulates knowledge on-line during interaction between the controller and the controlled system that improves control performance over time. Samuel's famous learning program for the game of checkers [47, 48], for example, improved its play by using a form of real-time DP based on a complete and accurate model of the game of checkers. However, we also devoted considerable attention to the adaptive version of Markovian decision problems in which an accurate model is not available. We described indirect and direct approaches to these problems. The method we called the *generic indirect method* is representative of the majority of algorithms described in the control theory literature. A system identification algorithm improves a system model on-line during control, and the controller determines each of its actions by executing a conventional DP algorithm at each time step based on the current system model. Although this approach is theoretically convenient, it is much too costly to apply to any but the smallest problems.

Adaptive real-time DP results from substituting real-time DP for conventional DP in the generic indirect method. This means that real-time DP is executed using the most recent system model generated by the system identification algorithm. Adaptive real-time DP can be tailored for the available computational resources by adjusting the number of DP stages it executes at each time step. Due to the additional uncertainty in the adaptive case, learning is necessarily slower than in the non-adaptive case when measured by the number of actions required. However, the amount of computation required to decide on each control action is roughly the same. This means that it is practical to apply adaptive real-time DP to problems that are much larger than those for which it is practical to apply methods, like the generic indirect method, that repeatedly execute a costly control design procedure.

In addition to indirect adaptive methods, we discussed direct adaptive methods. Direct methods do not form explicit models of the system underlying the decision problem. We described Watkin's [67] Q-learning algorithm, which approximates the optimal evaluation function without forming estimates of state-transition probabilities. Q-learning uses sample state transitions, either generated by a system model or observed in real-time, to produce a kind of Monte Carlo asynchronous DP. Following the logic by which we viewed asynchronous DP as an off-line DP algorithm whose stages are interleaved with control steps to produce real-time DP, we first presented Q-learning as an off-line algorithm.

Q-learning is a DP algorithm that is asynchronous at a finer grain than is asynchronous DP. Whereas the basic operation of asynchronous DP is backing up the cost of a state, which always requires minimizing over all admissible actions, the basic operation of Q-learning is updating the Q-value of a state-action pair, a computation less dependent on the number of admissible actions. The fine grain of the basic Q-learning update allows real-time Q-learning to focus on selected actions in addition to selected states in a way that is responsive to the observed behavior of the controlled system. The cost of this flexibility is the increased space complexity of Q-learning compared to adaptive real-time DP and the fact that the basic Q-learning update does not gather as much information as does backing up a state's cost when the state-transition probabilities are known, or good estimates are available.

Sophisticated exploration strategies are important in both non-adaptive and adaptive Markovian decision problems. In the non-adaptive case, an exploration strategy can improve control performance by decreasing the time required to reach goal states or, in the case of real-time DP, by focusing DP stages on states from which information most useful for improving the evaluation function is likely to be gained. Knowledgeable choice of the ordering of back ups can accelerate convergence of asynchronous DP, whether applied off- or on-line. In the adaptive case, exploration is also useful for these reasons, but exploration strategies must also address the necessity to gather information about the unknown structure of the system being controlled. Unlike exploration in the non-adaptive case, which can be conducted off-line based on the system model, this kind of exploration must be conducted on-line. We discussed how exploration performed for this reason conflicts with the performance objective of control, at least on a short-term basis, and that the controller should not always generate the actions that appear to be the best based on its current evaluation function.

Although we did not use sophisticated exploration strategies in our simulations of the race track problem, and we made no attempt in this article to analyze the difficult issues pertinent to exploration, sophisticated exploration strategies will play an essential role in making DP-based learning methods practical for larger problems. From what we did mention, however, it should be clear that it is not easy to devise a consistent set of desiderata for exploration strategies. For example, researchers have argued that an exploration strategy should 1) visit states in regions of the state space where information about the system is of low quality (to learn more about these regions), 2) visit states in regions of the state space where information about the system is of high quality (so that the back-up operation uses accurate estimates of the state-transition probabilities), or 3) visit states having successors whose costs are close to their optimal costs (so that the back-up operation efficiently propagates cost information). Each of these suggestions makes sense in the proper context, but it is not clear how to design a strategy that best incorporates all of them. It is encouraging, however, that the convergence results we have presented in this article are compatible with a wide range of exploration strategies.

Also critical in making these methods practical for large problems will be means for efficiently storing state costs or Q-values. Much of the research on DP-based learning methods has made use of storage schemes that do not use the simple lookup-table representation to which we have restricted attention. In problems in which DP-based learning algorithms focus on increasingly small subsets of states, as illustrated in our simulations of the race track problem, data structures such as hash tables and kd-trees can allow the algorithms to perform well despite the reduced space available. One can also adapt supervised learning procedures to use each back-up operation of a DP-based learning method to provide training information. If these methods can generalize adequately from the training data, they can provide efficient means for storing evaluation functions. Although some success has been achieved with methods that can generalize, such as connectionist networks, the theory we have presented in this article does not automatically extend to these cases. Generalization can disrupt the convergence of asynchronous DP. Additional research is needed to to understand how one can effectively combine function approximation methods with asynchronous DP.

Throughout this article we have assumed that the states of the system being controlled are completely and unambiguously observable by the controller. Although this assumption is critical to the theory and operation of all the algorithms we discussed, it can be very difficult to satisfy in practice. For example, the current state of a robot's world is vastly different from a list of the robot's current "sensations." On the positive side, effective closed-loop control policies do not have to distinguish between all possible sensations. However, exploiting this fact requires the ability to recognize states in the complex flow of sensations. Although the problem of state identification has been the subject of research in a variety of disciplines, and many approaches have been studied under many guises, it remains a critical factor in extending the applicability of DP-based learning methods. Any widely applicable approach to this problem must take the perspective that what constitutes a system's state for purposes of control—indeed what constitutes the system itself—is not independent of the control objectives. The framework adopted in this article in which a "dynamical system underlies a decision problem" is misleading in suggesting the existence of a single definitive grain with which to delineate events and to mark their passage. In actuality, control objectives dictate what is important in the flow of the controller's sensations, and multiple models at different levels of abstraction may be needed to achieve these objectives. If this caution is recognized, however, the algorithms described in this article should find wide application as components of sophisticated learning control systems.

Appendices

A Undiscounted Stochastic Optimal Path Problems with Improper Policies

A simple example shows that if there is an improper policy in a stochastic optimal path problem, then convergence of asynchronous DP requires the immediate costs to be positive. The example also illustrates some of the differences between synchronous and asynchronous DP. Figure 3 shows two states that are isolated from other states under some improper policy. This policy causes the transitions among these states shown by the arrows in the figure; state transitions for any other possible policies are not shown. The immediate costs for these states and the actions specified by the improper policy are the numbers on the arrows. The evaluation function for this improper policy is unbounded for states a and b because the net cost each time around the loop is +1. The undiscounted sum of these costs is infinite. Hence, if this improper policy were optimal, the stage-k costs $f_k(a)$ and $f_k(b)$ would grow without bound as k increases in undiscounted synchronous DP. But this cannot happen when a proper policy exists because synchronous DP will eventually find that the actions given by the proper policy yield lower costs than those given by the improper policy. Thus, synchronous DP will never converge to an improper policy when there is a proper policy, even if the immediate costs are not all positive.

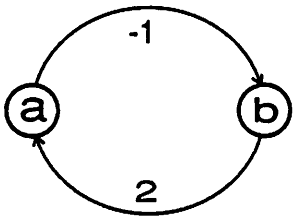

Figure 3: Two States Isolated Under Some Improper Policy. The immediate costs of these states and the actions specified by the improper policy are the numbers on the arrows. Synchronous and asynchronous DP can yield different costs for these states.

This need not be true for asynchronous DP. For example, suppose the asynchronous algorithm backs up a's cost exactly twice between every backup of b's cost. Then $f_k(b)$ will never change from the initial estimate, $f_0(b)$, while $f_k(a)$ will oscillate between $f_0(b) - 1$ and $f_0(b) - 2$. In this case, the improper policy giving rise to these costs may appear to be better than any proper policy so that the asynchronous algorithm will never discover the true least-cost actions for these states. This difficulty is not present when all the immediate costs are positive as shown in Appendix B.

B Asynchronous Solution of Undiscounted Stochastic Optimal Path Problems

The *Asynchronous Convergence Theorem* of Bertsekas and Tsitsiklis (ref. [9], p. 431) allows one to prove that asynchronous DP converges under quite general conditions. Specializing this theorem to the case of asynchronous DP, let T denote the operator implemented by a stage of the corresponding synchronous DP algorithm, that is,

$$f_{k+1} = T(f_k).$$

We use T^k to denote the operator resulting from the k-fold application of T, that is

$$T^k(f) = T(\ldots T(T(f)) \ldots),$$

where there are k Ts on the right. Suppose there is a sequence of nonempty sets $\{F_k\}$ of possible evaluation functions satisfying the following conditions:

1. $F_{k+1} \subseteq F_k$, for all $k = 0, 1, \ldots$,
2. $T(f) \in F_{k+1}$, for all $f \in F_k$ and all $k = 0, 1, \ldots$
3. If $\{g_k\}$ is a sequence such that $g_k \in F_k$ for every k, then every limit point of $\{g_k\}$ is a fixed point of T, and
4. for every k, there exist sets F_k^i, in this case of real numbers, for $i = 1, \ldots, n$, where n is the number of states, such that

$$F_k = F_k^0 \times F_k^1 \times \ldots \times F_k^n.$$

Then, if the initial evaluation function, f_0, of the asynchronous DP algorithm belongs to the set F_0, the algorithm converges to the optimal evaluation function, f^*.

We can use this result to prove the following theorem.

Theorem: An undiscounted stochastic optimal path problem can be solved by asynchronous DP provided that the following three conditions hold:

1. the initial cost of every goal state is zero,
2. there exists at least one proper policy, and
3. all immediate costs incurred by transitions from non-goal states are positive.

Proof: Define the functions $\underline{f} \equiv 0$ and

$$\bar{f}(i) = \begin{cases} 0 & \text{if } i \text{ is a goal state} \\ \infty & \text{otherwise.} \end{cases}$$

Because there exists at least one proper policy and the immediate costs are positive, we know that $\underline{f} \leq f^* \leq \bar{f}$.[19] Also observe that because all the immediate costs are positive, T is monotonic, i.e., $f \leq f' \Rightarrow T(f) \leq T(f')$.

By a convergence result for synchronous DP applied to undiscounted stochastic optimal path problems proved by Bertsekas and Tsitsiklis (ref. [9], Proposition 3.3, p. 318), we know that synchronous DP converges to f^* under the present assumptions (positive immediate costs imply that every improper policy must have at least one state whose cost is infinite, as required by Proposition 3.3).

We can now show that the conditions (listed above) required by the *Asynchronous Convergence Theorem* of Bertsekas and Tsitsiklis [9] are satisfied. Let

$$F_k = \left\{ f \mid T^k(\underline{f}) \leq f \leq T^k(\bar{f}) \right\}.$$

No set F_k is empty because $f^* \in F_0$ and f^* is a fixed point of T. Clearly if $f \in F_k$, then $T(f) \in F_{k+1}$. We also know that $F_{k+1} \subseteq F_k$ by the monotonicity of T. Because synchronous DP converges to f^*, we have that $\lim_{k\to\infty} T^k(\underline{f}) = f^*$ and that $\lim_{k\to\infty} T^k(\bar{f}) = f^*$. Therefore, if $\{g_k\}$ is any sequence with $g_k \in F_k$ for every k, then $\lim_{k\to\infty} g_k = f^*$, which is the unique fixed point of T. Finally, note that by the definition of F_k,

$$F_k = \left[(T^k\underline{f})(0), (T^k\bar{f})(0)\right] \times \left[(T^k\underline{f})(1), (T^k\bar{f})(1)\right] \times \ldots \times \left[(T^k\underline{f})(n), (T^k\bar{f})(n)\right].$$

Finally, note that setting the initial cost of every goal state to zero means that $f_0 \in F_0$. Q.E.D.

C Proof of the Trial-Based Real-Time DP Theorem

Here we prove the following theorem, which extends Korf's [32] convergence theorem for LRTA* to stochastic optimal path problems:

Theorem (Trial-Based Real-Time DP): In undiscounted stochastic optimal path problems, trial-based real-time DP, with the initial state of each trial restricted to a set of start states,

[19]For real-valued functions f and f' having the same domain S, $f \leq f'$ means that for all $i \in S$, $f(i) \leq f'(i)$.

ensures that the costs of all states that can be reached from any start state using an optimal policy converge with probability one to their optimal costs under the following conditions: 1) the initial cost of every goal state is zero, 2) there is at least one proper policy, 3) $c_i(u) > 0$ for all non-goal states i and actions $u \in U(i)$, and 4) $f_0(i) \leq f^*(i)$ for all states $i \in S$.

Proof: We first prove the theorem for the special case in which only the cost of the current state is backed up at each time interval, i.e., $B_t = \{s_t\}$ and $k_t = t$, for $t = 0, 1, \ldots$ (see Section 6.1). We then observe that the proof does not change when each B_t is allowed to be an arbitrary set containing s_t. Let G denote the goal set and let s_t , u_t, and f_t respectively denote the state, action, and evaluation function at time step t in an arbitrary infinite sequence of states, actions, and evaluation functions generated by trial-based real-time DP starting from an arbitrary start state.

First observe that the evaluation functions remain non-overestimating, i.e., at any time t, $f_t(i) \leq f^*(i)$ for all states i. This is true by induction because $f_{t+1}(i) = f_t(i)$ for all $i \neq s_t$ and if $f_t(j) \leq f^*(j)$ for all $j \in S$, then for all t

$$\begin{aligned} f_{t+1}(s_t) &= \min_{u \in U(i)} \left[c_{s_t}(u) + \sum_{j \in S} p_{s_t j}(u) f_t(j) \right] \\ &\leq \min_{u \in U(i)} \left[c_{s_t}(u) + \sum_{j \in S} p_{s_t j}(u) f^*(j) \right] = f^*(s_t), \end{aligned}$$

where the last equality restates the Bellman Optimality Equation (Equation 4).

Let $I \subseteq S$ be the set of all states that appear infinitely often in this arbitrary sequence; I must be nonempty because the state set is finite. Let $A(i) \in U(i)$ be the set of admissible actions for state i that have zero probability of causing a transition to a state not in I, i.e., $A(i)$ is the set of all actions $u \in U(i)$ such that $p_{ij}(u) = 0$ for all $j \in (S - I)$. Because states in $S - I$ appear a finite number of times, there is a finite time T_0 after which all states visited are in I. Then with probability one any action chosen an infinite number of times for any state i that occurs after T_0 must be in $A(i)$ (or else with probability one a transition out of I would occur), and so with probability one there must exist a time $T_1 \geq T_0$ such that for all $t > T_1$, we not only have that $s_t \in I$ but also that $u_t \in A(s_t)$.

We know that at each time step t, real-time DP backs up the cost of s_t because $s_t \in B_t$. We can write the back-up operation as follows:

$$f_{t+1}(s_t) = c_{s_t}(u_t) + \sum_{j \in I} p_{s_t j}(u_t) f_t(j) + \sum_{j \in (S-I)} p_{s_t j}(u_t) f_t(j). \tag{14}$$

But for all $t > T_1$, we know that $s_t \in I$ and that $p_{s_t j}(u_t) = 0$ for all $j \in S - I$. because $u_t \in A(s_t)$. Thus, for $t > T_1$ the right-most summation in Equation 14 is zero. This means that the costs of the states in $S - I$ have no influence on the operation of real-time DP after T_1. Thus, after T_1, real-time DP performs asynchronous DP on a Markovian decision problem with state set I.

If no goal states are contained in I, then all the immediate costs in this Markovian decision problem are positive. Because there is no discounting, it can be shown that asynchronous DP must cause the costs of the states in I to grow without bound. But this contradicts the fact that the cost of a state can never overestimate its optimal cost, which must be finite due to the existence of a proper policy. Thus I contains a goal state with probability one.

After T_1, therefore, trial-based real-time DP performs asynchronous DP on a stochastic optimal path problem with state set I that satisfies the conditions of the convergence theorem for asynchronous DP applied to undiscounted stochastic optimal path problems (Bertsekas and Tsitsiklis [9], Proposition 3.3, p. 318). Consequently, trial-based real-time DP converges to the optimal evaluation function of this stochastic optimal path problem. We also know that the optimal evaluation function for this problem is identical to the optimal evaluation function for the original problem restricted to the states in I because the costs of the states in $S - I$ have no influence on the costs of states in I after time T_1.

Furthermore, with probability one I contains the set of all states reachable from any start state via any optimal policy. Clearly, I contains all the start states because each start state begins an infinite number of trails. Trial-based real-time DP always executes a greedy action with respect to the current evaluation function and breaks ties in such a way that it continues to execute all the greedy actions. Because we know that the number of policies is finite and that trial-based real-time DP converges to the optimal evaluation function restricted to I, there is a time after which it continues to select all the actions that are greedy with respect to the optimal evaluation function, i.e., all the optimal actions. Thus with probability one I contains all the states reachable from any start state via any optimal policy.

Finally, with trivial revision the above argument holds if real-time DP backs up the costs of states other than the current state at each time step, i.e., if each B_t is an arbitrary subset of S.
Q.E.D.

D Simulation Details

Except for the discount factor, which we set to one throughout the simulations, real-time DP does not involve any parameters. Gauss-Seidel DP only requires specifying a state ordering for its sweeps. We selected an ordering without concern for any influence it might have on convergence rate. Both adaptive real-time DP and real-time Q-learning require exploration during the training trials, which we implemented using Equation 8. The parameter

T decreased with successive trials as follows:

$$\begin{aligned} T(0) &= T_{\text{Max}} \\ T(k+1) &= T_{\text{Min}} + \beta(T(k) - T_{\text{Min}}), \end{aligned}$$

where k is the trial number and $0 < \beta < 1$. For both algorithms, $T_{\text{Max}} = 75$ and $\beta = 0.992$. For adaptive real-time DP, $T_{\text{Min}} = 0.25$, and for real-time Q-learning, $T_{\text{Min}} = 0.5$.

Real-time Q-learning additionally requires sequences of learning rate parameters $\alpha_t(i, u)$ (Equation 11) that satisfy the hypotheses of the Q-Learning Theorem. We defined these sequences as follows. Let $\alpha_t(i, u)$ denote the learning rate parameter used when the Q-value of the state-action pair (i, u) is updated at time step t. Let $n_t(i, u)$ be the number of updates performed on the Q-value of (i, u) up to time step t. The learning rate $\alpha_t(i, u)$ is defined as follows:

$$\alpha_t(i, u) = \frac{\alpha_0 \tau}{\tau + n_t(i, u)}$$

where α_0 is the initial learning rate. We set $\alpha_0 = 0.5$ and $\tau = 300$. This equation implements a *search-then-converge* schedule for each $\alpha_t(i, u)$ as suggested by Darken and Moody [13]. They argue that such schedules can achieve good performance in stochastic optimization tasks. It can be shown that this schedule satisfies the hypotheses of the convergence theorem for off-line Q-Learning.

References

[1] C. W. Anderson. Strategy learning with multilayer connectionist representations. Technical Report TR87-509.3, GTE Laboratories, Incorporated, Waltham, MA, 1987. (This is a corrected version of the report published in *Proceedings of the Fourth International Workshop on Machine Learning*,103–114, 1987, San Mateo, CA: Morgan Kaufmann.).

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:835–846, 1983. Reprinted in J. A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, MA, 1988.

[3] A. G. Barto, R. S. Sutton, and C. Watkins. Sequential decision problems and neural networks. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 686–693, San Mateo, CA, 1990. Morgan Kaufmann.

[4] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins. Learning and sequential decision making. In M. Gabriel and J. Moore, editors, *Learning and Computational Neuroscience:Foundations of Adaptive Networks*, pages 539–602. MIT Press, Cambridge, MA, 1990.

[5] A.G. Barto and S.P. Singh. On the computational economics of reinforcement learning. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Connectionist Models Proceedings of the 1990 Summer School*, pages 35–44. Morgan Kaufmann, San Mateo, CA, 1991.

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

[7] D. P. Bertsekas. Distributed dynamic programming. *IEEE Transactions on Automatic Control*, 27:610–616, 1982.

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

[9] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[10] R. Boundarel, J. Delmas, and P. Guichet. *Dynamic Programming and its Application to Optimal Control.* Academic Press, New York, 1971.

[11] D. Chapman. Penquins can make cake. *AI Magazine*, 10:45–50, 1989.

[12] D. Chapman and L. P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the 1991 International Joint Conference on Artificial Intelligence.* To appear.

[13] C. Darken and J. Moody. Note on learning rate schedule for stochastic optimization. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 832–838, San Mateo, CA, 1991. Morgan Kaufmann.

[14] P. Dayan. Navigating through temporal difference. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 464–470, San Mateo, CA, 1991. Morgan Kaufmann.

[15] P. Dayan. *Reinforcing Connectionism: Learning the Statistical Way.* PhD thesis, University of Edinburgh, 1991.

[16] M. Gardner. Mathematical games. *Scientific American*, 228:108, January 1973.

[17] M. L. Ginsberg. Universal planning: An (almost) universally bad idea. *AI Magazine*, 10:40–44, 1989.

[18] R. Gonzalez and E. Rofman. On deterministic control problems: An approximate procedure for the optimal cost I. The stationary problem. *SIAM J. Control and Optimization*, 23:242–266, 1985.

[19] G. C. Goodwin and K. S. Sin. *Adaptive Filtering Prediction and Control.* Prentice-Hall, Englewood Cliffs, N.J., 1984.

[20] V. Gullapalli. A comparison of supervised and reinforcement learning methods on a reinforcement learning task. In *Proceedings of the 1991 IEEE Symposium on Intelligent Control.* Arlington, VA. To appear.

[21] S. E. Hampson. *Connectionist Problem Solving: Computational Aspects of Biological Learning.* Birkhauser, Boston, 1989.

[22] J. H. Holland. Escaping brittleness: The possibility of general-purpose learning algorithms applied to rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II,* pages 593–623. Morgan Kaufmann, San Mateo, CA, 1986.

[23] D. H. Jacobson and D. Q. Mayne. *Differential Dynamic Programming.* Elsevier, New York, 1970.

[24] A. Jalali and M. Ferguson. Computationally efficient adaptive control algorithms for Markov chains. In *Proceedings of the 28th Conference on Decision and Control,* pages 1283–1288, Tampa, Florida, 1989.

[25] M. I. Jordan and R. A. Jacobs. Learning to control an unstable system with forward modeling. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2,* San Mateo, CA, 1990. Morgan Kaufmann.

[26] L. P. Kaelbling. *Learning in Embedded Systems.* PhD thesis, Stanford University, Department of Computer Science, Stanford, CA, 1990. Technical Report TR-90-04.

[27] L. P. Kaelbling. *Learning in Embedded Systems.* MIT Press, Cambridge, MA, 1991. Revised version of Teleos Research TR-90-04, June 1990.

[28] R. L. Kasyap, C. C. Blaydon, and K. S. Fu. Stochastic approximation. In J. M. Mendel and K. S. Fu, editors, *Adaptive, Learning, and Pattern Recognition Systems: Theory and Applications.* Academic Press, New York, 1970.

[29] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science,* 220:671–680, 1983.

[30] A. H. Klopf. Brain function and adaptive systems—A heterostatic theory. Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA, 1972. A summary appears in *Proceedings of the International Conference on Systems, Man, and Cybernetics,* 1974, IEEE Systems, Man, and Cybernetics Society, Dallas, TX.

[31] A. H. Klopf. *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence.* Hemishere, Washington, D.C., 1982.

[32] R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[33] P. R. Kumar. A survey of some results in stochastic adaptive control. *SIAM Journal of Control and Optimization*, 23:329–380, 1985.

[34] V. Kumar and L. N. Kanal. The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In L. N. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, pages 1–37. Springer-Verlag, 1988.

[35] W. H. Kwon and A. E. Pearson. A modified quadratic cost problem and feedback stabilization of a linear system. *IEEE Transactions on Automatic Control*, 22:838–842, 1977.

[36] Y. le Cun. A theoretical framework for back-propagation. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28. Morgan Kaufmann, San Mateo, CA, 1988.

[37] M. Lemmon. Real-time optimal path planning using a distributed computing paradigm. In *Proceedings of the American Control Conference*, Boston, MA, 1991.

[38] Long-Ji Lin. Self-improvement based on reinforcement learning, planning and teaching. In L. A. Birnbaum and G. C. Collins, editors, *Maching Learning: Proceedings of the Eighth International Workshop*, pages 323–327, San Mateo, CA, 1991. Morgan Kaufmann.

[39] Long-Ji Lin. Self-improving reactive agents: Case studies of reinforcement learning frameworks. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 297–305, Cambridge, MA, 1991. MIT Press.

[40] D. Q. Mayne and H. Michalska. Receding horizon control of nonlinear systems. *IEEE Transactions on Automatic Control*, 35:814–824, 1990.

[41] D. Michie and R. A. Chambers. BOXES: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence 2*, pages 137–152. Oliver and Boyd, 1968.

[42] M. L. Minsky. *Theory of Neural-Analog Reinforcement Systems and its Application to the Brain-Model Problem.* PhD thesis, Princeton University, 1954.

[43] M. L. Minsky. Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers*, 49:8–30, 1961. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought.* McGraw-Hill, New York, 406–450, 1963.

[44] A. W. Moore. *Efficient Memory-Based Learning for Robot Control.* PhD thesis, University of Cambridge, Cambridge, UK, 1990.

[45] A. W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In L. A. Birnbaum and G. C. Collins, editors, *Maching Learning: Proceedings of the Eighth International Workshop*, pages 333–337, San Mateo, CA, 1991. Morgan Kaufmann.

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

[47] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, pages 210–229, 1959. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York, 1963.

[48] A. L. Samuel. Some studies in machine learning using the game of checkers. II—Recent progress. *IBM Journal on Research and Development*, pages 601–617, November 1967.

[49] J. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Institut für Informatik, Technische Universität München, Arcisstr. 21, 800 München 2, Germany, 1991.

[50] J. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 222–227, Cambridge, MA, 1991. MIT Press.

[51] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1039–1046, Menlo Park, CA, 1987.

[52] M. J. Schoppers. In defense of reaction plans as caches. *AI Magazine*, 10:51–60, 1989.

[53] S. P. Singh. Transfer of learning across compositions of sequential tasks. In L. A. Birnbaum and G. C. Collins, editors, *Maching Learning: Proceedings of the Eighth International Workshop*, pages 348–352, San Mateo, CA, 1991. Morgan Kaufmann.

[54] S.P. Singh. Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning*, to appear.

[55] H. Stephanou, editor. *Special Issue on Intelligent Control.* IEEE Control Systems Magazine, June 1991. 11.

[56] R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[57] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

[58] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, San Mateo, CA, 1990. Morgan Kaufmann.

[59] R. S. Sutton. Planning by incremental dynamic programming. In L. A. Birnbaum and G. C. Collins, editors, *Maching Learning: Proceedings of the Eighth International Workshop*, pages 353–357, San Mateo, CA, 1991. Morgan Kaufmann.

[60] R. S. Sutton and A. G. Barto. Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review*, 88:135–170, 1981.

[61] R. S. Sutton and A. G. Barto. A temporal-difference model of classical conditioning. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, 1987. Erlbaum.

[62] R. S. Sutton and A. G. Barto. Time-derivative models of Pavlovian reinforcement. In M. Gabriel and J. Moore, editors, *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pages 497–537. MIT Press, Cambridge, MA, 1990.

[63] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. In *Proceedings of the American Control Conference*, pages 2143–2146, Boston, MA, 1991.

[64] M. Tan. Learning a cost-sensitive internal representation for reinforcement learning. In L. A. Birnbaum and G. C. Collins, editors, *Maching Learning: Proceedings of the Eighth International Workshop*, pages 358–362, San Mateo, CA, 1991. Morgan Kaufmann.

[65] S. B. Thrun and K. Möller. Active exploration in dynamic environments. Submitted for publication.

[66] P. E. Utgoff and J. A. Clouse. Two kinds of training information for evaluation function learning. In *Proceedings of the Ninth Annual Conference on Artificial Intelligence*, pages 596–600, San Mateo, CA, 1991. Morgan Kaufmann.

[67] C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, Cambridge, England, 1989.

[68] C. J. C. H. Watkins and P. Dayan. Q-learning. Submitted for publication.

[69] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[70] P. J. Werbos. Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook*, 22:25–38, 1977.

[71] P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In R. F. Drenick and F. Kosin, editors, *System Modeling an Optimization*. Springer-Verlag, 1982. Proceedings of the Tenth IFIP Conference, New York, 1981.

[72] P. J. Werbos. Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, 1987.

[73] P. J. Werbos. Generalization of back propagation with applications to a recurrent gas market model. *Neural Networks*, 1:339–356, 1988.

[74] P. J. Werbos. Neural networks for control and system identification. In *Proceedings of the 28th Conference on Decision and Control*, pages 260–265, Tampa, Florida, 1989.

[75] P. J. Werbos. Consistency of HDP applied to simple reinforcement learning problem. *Neural Networks*, 3:179–189, 1990.

[76] R. M. Wheeler and K. S. Narendra. Decentralized learning in finite Markov chains. *IEEE Transactions on Automatic Control*, 31:519–526, 1986.

[77] S. D. Whitehead. Complexity and cooperation in Q-learning. In L. A. Birnbaum and G. C. Collins, editors, *Maching Learning: Proceedings of the Eighth International Workshop*, pages 363–367, San Mateo, CA, 1991. Morgan Kaufmann.

[78] S. D. Whitehead and D. H. Ballard. A study of cooperative mechanisms for faster reinforcement learning. Technical Report TR 365, University of Rochester, Computer Science Department, 1991.

[79] R. J. Williams and L. C. Baird, III. A mathematical analysis of actor-critic architectures for learning optimal controls through incremental dynamic programming. In *Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems*, pages 96–101, New Haven, CT, Aug 1990.

[80] I. H. Witten. An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34:286–295, 1977.

[81] I. H. Witten. Exploring, modelling and controlling discrete sequential environments. *International Journal of Man-Machine Studies*, 9:715–735, 1977.

[82] L. E. Wixsom. Scaling reinforcement learning techniques via modularity. In L. A. Birnbaum and G. C. Collins, editors, *Maching Learning: Proceedings of the Eighth International Workshop*, pages 368–372, San Mateo, CA, 1991. Morgan Kaufmann.