Chapter 1
Introduction

In the industrial age, humans delegated physical labor to machines. Now, in the information age, we are increasingly delegating *mental* labor, charging computers with such tasks as controlling traffic signals, scheduling factory production, planning medical treatments, allocating investment portfolios, routing data through communications networks, and even playing expert-level backgammon or chess. Such tasks are difficult *sequential decision problems*:

- the task calls not for a single decision, but rather for a whole series of decisions over time;
- the outcome of any decision may depend on random environmental factors beyond the computer's control; and
- the ultimate objective—measured in terms of traffic flow, patient health, business profit, or game victory—depends in a complicated way on many interacting decisions and their random outcomes.

In such complex problems, optimal decision policies are in general unknown, and it is often difficult, even for human domain experts, to manually encode even reasonably good decision policies in software. A growing body of research in Artificial Intelligence suggests the following alternative methodology:

> **A decision-making algorithm can autonomously *learn* effective policies for sequential decision tasks, simply by simulating the task and keeping statistics on which decisions tend to lead to good ultimate performance and which do not.**

The field of *reinforcement learning*, to which this thesis contributes, defines a principled foundation for this methodology.

1.1 Motivation: Learning Evaluation Functions

In Artificial Intelligence, the fundamental data structure for decision-making in large state spaces is the *evaluation function*. Which state should be visited next in the search for a better, nearer, cheaper goal state? The evaluation function maps features of each state to a real value that assesses the state's promise. For example, in the domain of chess, a classic evaluation function is obtained by summing material advantage weighted by 1 for pawns, 3 for bishops and knights, 5 for rooks, and 9 for queens. The choice of evaluation function "critically determines search results" [Nilsson 80, p.74] in popular algorithms for planning and control (A^*), game-playing (alpha-beta), and combinatorial optimization (hillclimbing, simulated annealing).

Evaluation functions have generally been designed by human domain experts. The weights {1,3,3,5,9} in the chess evaluation function given above summarize the judgment of generations of chess players. IBM's Deep Blue chess computer, which defeated world champion Garry Kasparov in a 1997 match, used an evaluation function of over 8000 tunable parameters—the values of which were set initially by an automatic procedure, but later carefully hand-tuned under the guidance of a human grandmaster [Hsu *et al.* 90, Campbell 98]. Similar tuning occurs in combinatorial optimization domains such as the Traveling Salesperson Problem [Lin and Kernighan 73] and VLSI circuit design tasks [Wong *et al.* 88]. In such domains the state space consists of legal candidate solutions, and the domain's *objective function*—the function that evaluates the quality of a final solution—can itself serve as an evaluation function to guide search. However, if the objective function has many local optima or regions of constant value (plateaus) with respect to the available search moves, then it will not be effective as an evaluation function. Thus, to get good optimization results, engineers often spend considerable effort tweaking the coefficients of penalty terms and other additions to their objective function; I cite several examples of this in Chapter 3. Clearly, automatic methods for building evaluation functions offer the potential both to save human effort and to optimize search performance more effectively.

1.2 The Promise of Reinforcement Learning

Reinforcement learning (RL) provides a solid foundation for learning evaluation functions for sequential decision problems. Standard RL methods assume that the problem can be formalized as a Markov decision process (MDP), a model of controllable dynamic systems used widely in control theory, artificial intelligence, and operations research [Puterman 94]. I describe the MDP model in detail in Chapter 2. The key fact about this model is that for any MDP, there exists a special evaluation function known as the *optimal value function*. Denoted by $V^*(x)$, the optimal value function predicts the expected long-term reward available from each state x when all future decisions are made optimally. V^* is an ideal evaluation function: a greedy one-step lookahead search based on V^* identifies precisely the optimal long-term decision to make at each state. The problem, then, becomes how to compute V^*.

Algorithms for computing V^* are well understood in the case where the MDP state space is relatively small (say, fewer than 10^7 discrete states), so that V^* can be implemented as a lookup table. In small MDPs, if we have access to the transition model which tells us the distribution of successor states that will result from applying a given action in a given state, then V^* may be calculated exactly by a variety of classical algorithms such as dynamic programming or linear programming [Puterman 94]. In small MDPs where the explicit transition model is not available, we must build V^* from sample trajectories generated by direct interaction with a simulation of the process; in this case, recently discovered reinforcement learning methods such as TD(λ) [Sutton 88], Q-learning [Watkins 89], and Prioritized Sweeping [Moore and Atkeson 93] apply. These algorithms apply dynamic programming in an asynchronous, incremental way, but under suitable conditions can still be shown to converge to V^* [Bertsekas and Tsitsiklis 96, Littman and Szepesvári 96].

The situation is very different for large-scale decision tasks, such as the transportation and medical domains mentioned at the start of this chapter. These tasks have high-dimensional state spaces, so enumerating V^* in a table is intractable—a problem known as the "curse of dimensionality" [Bellman 57]. One approach to escaping this curse is to approximate V^* compactly using a function approximator such as linear regression or a neural network. The combination of reinforcement learning and function approximation, known as *neuro-dynamic programming* [Bertsekas and Tsitsiklis 96] or *value function approximation* [Boyan *et al.* 95], has produced several notable successes on such problems as backgammon [Tesauro 92, Boyan 92], job-shop scheduling [Zhang and Dietterich 95], and elevator control [Crites and Barto 96]. However, these implementations are extremely computationally intensive, requiring many thousands or even millions of simulated trajectories to reach top performance. Furthermore, when general function approximators are used instead of lookup tables, the convergence proofs for nearly all dynamic programming and reinforcement learning algorithms fail to carry through [Boyan and Moore 95, Bertsekas 95, Baird 95, Gordon 95]. Perhaps the strongest convergence result for value function approximation to date is the following [Tsitsiklis and Roy 96]: for an MDP with a *fixed* decision-making policy, the TD(λ) algorithm may be used to calculate an accurate *linear* approximation to the value function. Though its assumption of a fixed policy is quite limiting, this theorem nonetheless applies to the learning done by STAGE, a practical algorithm for global optimization introduced in this dissertation.

1.3 Outline of the Dissertation

This thesis aims to advance the state of the art in value function approximation for large, practical sequential decision tasks. It addresses two questions:

1. Can we devise new methods for value function approximation that are robust and efficient?

2. Can we apply the currently available convergence results to practical problems?

Both questions are answered in the affirmative:

1. I discuss three new algorithms for value function approximation, which apply to three different restricted classes of Markov decision processes: Grow-Support for large deterministic MDPs (§2.2), ROUT for large acyclic MDPs (§2.3), and Least-Squares TD(λ) for large Markov chains (§6.1). Each is designed to gain robustness and efficiency by exploiting the restricted MDP structure to which it applies.

2. I introduce **STAGE**, a new reinforcement learning algorithm designed specifically for large-scale global optimization tasks. In STAGE, commonly applied local optimization algorithms such as stochastic hillclimbing are viewed as inducing fixed decision policies on an MDP. Given that view, TD(λ) or supervised learning may be applied to learn an approximate value function for the policy. STAGE then exploits the learned value function to improve optimization performance in real time.

The thesis is organized as follows:

Chapter 2 presents formal definitions and notation for Markov decision processes and value function approximation. It then summarizes Grow-Support and ROUT, algorithms which learn to approximate V^* in deterministic and acyclic MDPs, respectively. Both these algorithms build V^* strictly backward from the goal, even when given only a forward simulation model, as is usually the case. These algorithms have been presented previously [Boyan and Moore 95, Boyan and Moore 96], but this chapter offers a new unified discussion of both algorithms and new results and analysis for ROUT.

Chapter 3 introduces STAGE, the algorithm which is the main contribution of this dissertation [Boyan and Moore 97, Boyan and Moore 98]. STAGE is a practical method for applying value function approximation to arbitrary large-scale global optimization problems. This chapter motivates and describes the algorithm and discusses issues of theoretical soundness and computational efficiency.

Chapter 4 presents empirical results with STAGE on seven large-scale optimization domains: bin-packing, channel routing, Bayes net structure-finding, radiotherapy treatment planning, cartogram design, Boolean formula satisfiability, and Boggle board setup. The results show that on a wide range of problems, STAGE learns efficiently, effectively, and with minimal need for problem-specific parameter tuning.

Chapter 5 analyzes STAGE's success, giving evidence that reinforcement learning is indeed responsible for the observed improvements in performance. The sensitivity of the algorithm to various user choices, such as the feature representation and function approximator, and to various algorithmic choices, such as when to end a trial and how to begin a new one, is tested empirically.

Chapter 6 offers two significant investigations beyond the basic STAGE algorithm. In Section 6.1, I describe a least-squares implementation of TD(λ), which generalizes both standard supervised linear regression and earlier results on least-squares TD(0) [Bradtke and Barto 96]. In Section 6.2, I discuss ways of transferring knowledge learned by STAGE from already-solved instances to novel similar instances, with the goal of saving training time.

Chapter 7 reviews the relevant work from the optimization and AI literatures, situating STAGE at the confluence of adaptive multi-start local search methods, reinforcement learning methods, genetic algorithms, and evaluation function learning techniques for game-playing and problem-solving search.

Chapter 8 concludes with a summary of the thesis contributions and a discussion of the many directions for future research in value function approximation for optimization.