# On Integrating Apprentice Learning and Reinforcement Learning

Jeffery A. Clouse
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003 USA

Technical Report 97-26

# ON INTEGRATING APPRENTICE LEARNING AND REINFORCEMENT LEARNING

A Dissertation Presented

by

JEFFERY A. CLOUSE

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1996

Department of Computer Science

# ON INTEGRATING APPRENTICE LEARNING AND REINFORCEMENT LEARNING

A Dissertation Presented

by

JEFFERY A. CLOUSE

Approved as to style and content by:

Paul E. Utgoff, Chair

Andrew G. Barto, Member

Roderic A. Grupen, Member

John W. Donahoe, Member

David W. Stemple, Department Chair
Department of Computer Science

*This dissertation is dedicated to*

*Michele, Christopher, Andrew, and Patrick.*

# ACKNOWLEDGEMENTS

There are many people to thank for contributing to this dissertation and to my success in graduate school. Foremost is Professor Paul Utgoff, who has been my principal guide throughout my graduate career. This dissertation would not have been possible without Paul's patience and insight. I have benefited enormously from his scientific curiosity, attention to detail, and research standards, and can not thank him enough for all he has done.

I also wish to thank my committee, Andrew Barto, Roderic Grupen, and John Donahoe, for their interest in my work and for their help with the dissertation. Andy deserves special consideration for allowing me to give several seminars in his Reinforcement Learning classes.

The many denizens of the Machine Learning Lab, both past and present, are also due credit for my work. Carla Brodley, Jamie Callan, Tom Fawcett, Neil Berkman, Doina Precup, and Rodger Zanny have all provided me with helpful comments, reading my papers and sitting through many practice talks. Their continued comradeship has made graduate school bearable. I am also grateful to Gunnar Blix, an adopted member of our lab, for sharing his technical expertise in various subjects, and for his insights into my work, which have helped me clarify my own thoughts about it.

Many other individuals at the University of Massachusetts at Amherst have provided me with friendship, inspiration, and aid. Particularly dear to me are the members of my "Thesis Support Group," Joe McCarthy, Dorothy Mammen, Malini Bhandaru, and Bob Crites, with whom I shared the ups and downs of the dissertation process. Without their help I would still be stumbling along. I also thank Margie

Connell, Cris Pedregal-Martin, Jody Daniels, Zack Rubinstein, David Jensen, and Marty Humphrey for their advice and support; and I am indebted to many of the Computer Science staff for making my problems their problems.

This research was supported by the National Science Foundation under Grant No. IRI-9222766, and by the National Aeronautics and Space Administration under Grant No. NCC 2-658.

The most important contributors to my success at UMass, and without whom I would never have completed this dissertation, are my family. The faith that my parents and in-laws have in me has inspired me to do the best I can. My children, Christopher, Andrew, and Patrick, reminded me that there is a world outside of graduate school in which beauty and adventure abound, and were always happy to greet me with hugs and kisses after a long day in the office. Finally, I cannot express fully my gratitude to Michele, whose sacrifices, support, and encouragement over the years I will never be able to repay. I can only say, "Thank you for everything!"

# ABSTRACT

ON INTEGRATING
APPRENTICE LEARNING AND REINFORCEMENT LEARNING

SEPTEMBER 1996

JEFFERY A. CLOUSE

B.S., VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Paul E. Utgoff

Apprentice learning and reinforcement learning are methods that have each been developed in order to endow computerized agents with the capacity to learn to perform multiple-step tasks, such as problem-solving tasks and control tasks. To achieve this end, each method takes differing approaches, with disparate assumptions, objectives, and algorithms. In apprentice learning, the autonomous agent tries to mimic a training agent's problem-solving behavior, learning based on examples of the trainer's action choices. In an attempt to learn to perform its task optimally, the learner in reinforcement learning changes its behavior based on scalar feedback about the consequences of its own actions.

We demonstrate that a careful integration of the two learning methods can produce a more powerful method than either one alone. An argument based on the characteristics of the individuals maintains that a hybrid will be an improvement because of the complimentary strengths of its constituents. Although existing hybrids

of apprentice learning and reinforcement learning perform better than their individual components, those hybrids have left many questions unanswered. We consider the following questions in this dissertation. How do the learner and trainer interact during training? How does the learner assimilate the trainer's expertise? How does the proficiency of the trainer affect the learner's ability to perform the task? And, when during training should the learner acquire information from the trainer? In our quest for answers, we develop the ASK FOR HELP integrated approach, and use it in our empirical study.

With the new integrated approach, the learning agent is significantly faster at learning to perform optimally than learners employing either apprentice learning alone or reinforcement learning alone. The study indicates further that the learner can learn to perform optimally even when its trainer cannot; thus, the learner can outperform its trainer. Two strategies for determining when to acquire the trainer's aid show that simple approaches work well. The results of the study demonstrate that the ASK FOR HELP approach is effective for integrating apprentice learning and reinforcement learning, and support the conclusion that an integrated approach can be better than its individual components.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

This dissertation is about integrating two learning methods, each of which was developed to allow intelligent, computerized agents to learn to solve problems. In the first method, *apprentice learning*, the autonomous agent learns based on examples of another agent's problem-solving performance. In the other method, *reinforcement learning*, the agent changes its performance based on scalar feedback about the consequences of its own actions. The two methods each have different objectives, employ different algorithms, and make different assumptions in order to endow the agent with the capacity to learn.

In this dissertation we explore issues in integrating these two methods. The research is motivated mainly by two observations. First, we are inspired by human learners, who seem to employ a combination of the two approaches, refining their problem-solving skills autonomously and also adapting based on their observation of others. Second, previous research has identified this area as promising for building automated agents that can learn to solve problems. Hybrids of apprentice learning and reinforcement learning have been shown to perform better than their constituents, although such research has left many questions unanswered. Our main goal in this dissertation is to answer fundamental questions about integrated approaches, so that the approaches are understood better and can achieve wider applicability.

In the remainder of this chapter, we introduce the notion of a multiple-step task as a type of problem to solve, and discuss what is required in learning such a task. We give a quick introduction to the two individual learning methods, provide further

motivation for integrating them, and outline our particular approach. Finally, we give an overview of the results of the dissertation.

## 1.1 Multiple-Step Tasks

This dissertation focuses on computerized agents that learn to make sequences of decisions in order to solve problems. Examples of problems that require multiple decisions, also called multiple-step tasks (Dietterich, London, Clarkson & Dromey, 1982), include robot navigation, game-playing, and theorem proving. In robot navigation, the robot makes a sequence of decisions to arrive at its destination. The game-playing agent must select a series of moves that will ensure a win. And, a sequence of mathematical manipulations are required to prove a theorem.

Multiple-step tasks are specified in terms of *states* and *actions*. The states are the situations that arise in performing the task, such as the configurations of pieces on the chess board, and the actions, such as the legal piece moves, transform the task from state to state. Thus, the states and operators are abstract representations of the actual task, which is situated in a real environment. Figure 1.1 shows part of a solution to a multiple-step task. The circles in the figure represent the states, the arrows depict the actions, and the highlighted arrows represent the actions chosen by the agent. From the initial state, the agent first chose "a2," which produced "State A." The problem was then transformed to "State B" by the agent's choice of "a4." Through the continued selection of actions, the agent solves the problem.

As the agent performs the task, it may incur costs or receive pay-backs when applying actions. These costs, which are real-valued scalars that are also called *weights* and *rewards*, represent such concepts as distance between a state and its successor or the actual cost of performing an action. The objective of an agent solving the problem is to find a sequence of actions that optimizes a cumulative measure of the rewards. For example, the agent may attempt to find the shortest or least-cost path in solving

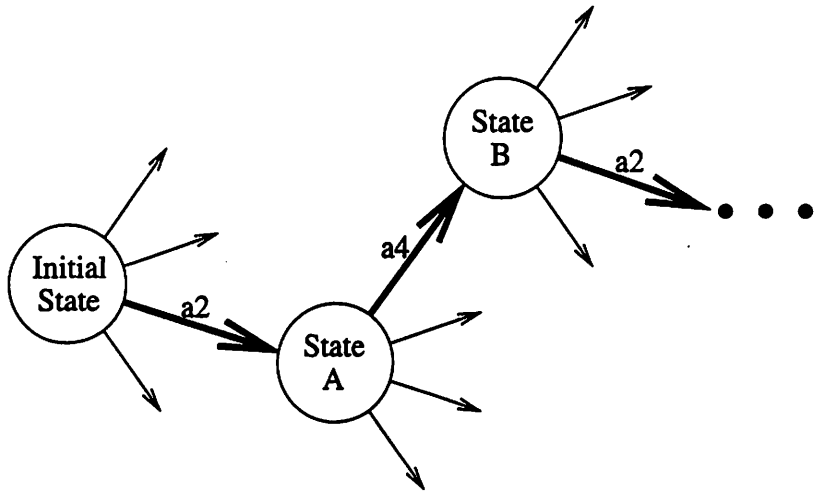



**Figure 1.1** A multiple-step task

the problem. Problems that simply require the agent to reach a specially designated goal state may also be expressed in terms of rewards by giving the transitions into the goal state positive weight and all other transitions zero weight.

## 1.2 Developing a Policy

At each discrete time step the agent faces the situation depicted in Figure 1.2. Namely, the agent observes the current state of the problem and must choose one of a set of actions to apply. To make this decision, the agent relies on its policy, a mapping from states to actions, which indicates which of the applicable actions should be chosen in order to complete the task successfully. The agent can develop its policy via exhaustive computation or limited-depth search, both of which have deficiencies. Fortunately, the shortcomings of these approaches can be overcome by having the agent *learn* its policy.

The policy that allows the agent to solve the problem can be determined by extensive computation. The agent can develop its policy by performing an exhaustive search in the space of solution paths, simulating the application of every action to ev-

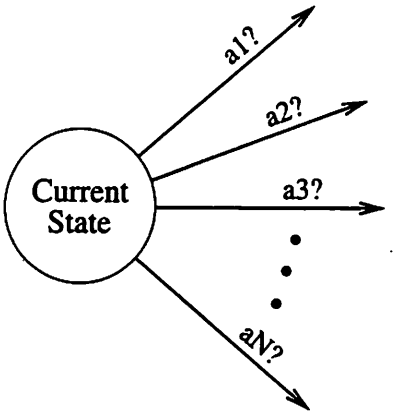



**Figure 1.2** Which action should be chosen?

ery state for every solution path. The agent might also develop its policy via dynamic programming (Bellman, 1957), an iterative technique for determining the agent's correct action at every state. Unfortunately, these two approaches are often impractical. First, they each usually require a complete specification of the problem in the form of correct reward and state-transition functions. Such functions can only be developed for very simple problems because any uncertainty about the problem precludes having a correct specification. Second, the computations required are often prohibitive. Finding a policy for even moderate-sized problems is typically intractable.

In order to reduce the computational complexity identified above, several systems implement the policy as various forms of limited-depth search (Nilsson, 1969; Slagle & Dixon, 1969; Rosenbloom, 1982; Berliner, 1984; Pearl, 1984; Korf, 1988). In these systems, a human designer attempts to capture each state's worthiness with respect to the problem-solving effort by codifying the designer's limited domain knowledge in a function that maps states to real-valued numbers, known as an evaluation function. Then, instead of performing an exhaustive search of the state-space to make an action choice, the agent chooses the action that produces the successor state with the highest value, as determined by limited search and application of the evaluation

function to the states at the frontier of the search. Regardless of the details of the particular search algorithm, these approaches are unattractive because their success depends heavily on the accuracy of the evaluation function. Developing a good evaluation function involves an expensive iterative process in which the human refines the function based on detailed analyses of the agent's inability to solve the problem. Moreover, these search techniques require a complete specification of the problem in order to perform any search at all.

Having the agent learn its policy can overcome the shortcomings of the approaches mentioned above. The learning agent does not need a complete description of the problem and, further, can develop a policy without performing computationally expensive operations. Moreover, learning supersedes the human's knowledge-intensive tweaking of an evaluation function because the agent has the ability to improve its policy over time, as it gains more knowledge about the problem.

## 1.3 Apprentice Learning and Reinforcement Learning

Early in the history of Artificial Intelligence, researchers began studying automated agents that learn to solve multiple-step tasks (Samuel, 1963; Samuel, 1967; Waterman, 1970; Fikes, Hart & Nilsson, 1972). The objective of the agents is to use the resources they have available in order to become more proficient at performing their tasks. Two methods in particular have come out of that effort: apprentice learning and reinforcement learning. Although both of these methods endow the agent with the capacity to learn to solve problems, they each take distinct approaches, with different assumptions, algorithms, and objectives.

In apprentice learning, the automated agent observes a training agent solving a problem. For each state and associated action in the trainer's solution, which are either provided to the learner or observed by the learner, the learner infers that the action was appropriate to perform in that state. The learner might further assume

that all other action choices were not applicable because the trainer did not pick them[1]. Accordingly, apprentice learning is a supervised learning technique where the training examples are derived from the trainer's solution. The learner's objective is to develop a good policy that can be applied to performing the problem. The learner attempts to meet this goal by acquiring the trainer's observed strategy. Thus, apprentice learning depends heavily on the trainer's expertise. This method has been employed successfully to learn such complex tasks as game-playing (Samuel, 1963), and vehicle control (Pomerleau, 1991; Sammut, Hurst, Kedzier & Michie, 1992).

In reinforcement learning, the automated agent adapts based on its own problem-solving experience, updating its policy via a sequence of trial-and-error experiments. That is, the agent learns entirely from its own decisions. At each discrete step in the problem, the agent determines its current action, performs that action, and observes the real-valued reward it receives. The rewards indicate how well the agent is performing the task, and the objective is to learn to select actions that optimize a measure of those rewards. Stated differently, the objective is to learn an optimal policy. The agent is faced with the difficult problem of credit assignment (Minsky, 1963): It must determine which of its many actions should be credited, or blamed, for the current reward, especially considering that the rewards may be sparse. Furthermore, the agent must trade off performing well on the task versus experimenting with its options in order to learn more about the task. Similarly to apprentice learning, many complex tasks have been tackled successfully with reinforcement learning, including game-playing (Samuel, 1967; Tesauro, 1995), robotics tasks (Gullapalli, 1991; Mahadevan & Connell, 1992; Singh, 1994), and elevator dispatching (Crites & Barto, 1996).

[1]Care must be taken when assuming that only the chosen action is correct because there may be situations in which many of the choices are equally correct.

## 1.4 Integrating the Two Learning Methods

In this dissertation, we investigate particular issues that arise in integrating apprentice learning and reinforcement learning. Although these two methods approach the difficulty of improving the agent's ability to solve problems from different perspectives, they each have desirable characteristics that make them viable. We argue that the integration of the two methods can produce a new learning method that combines the beneficial qualities of the two individuals and ameliorates their weaknesses, leading to improvements over each.

This viewpoint has been substantiated partially by previous research in building integrated systems. Utgoff and Clouse (1991) develop an integrated system in which the agent requires only one training trial to learn to perform its task, versus needing two trials when learning from only a training agent and requiring close to five hundred trials when learning via reinforcement learning. Other work (Clouse & Utgoff, 1992; Lin, 1992; Lin, 1993) shows that allowing a learning agent to incorporate a trainer's knowledge improves the speed with which the agent learns to perform the task greatly. Each of these systems shows the benefits of integrating apprentice learning and reinforcement learning and serves as inspiration for our work.

Although each of these systems achieves success via a combination of the two individual learning methods, they leave many questions unanswered. Our objective in this dissertation is to examine fundamental questions about the integration of apprentice learning and reinforcement learning. We investigate how the learner and trainer interact, how the learner incorporates the trainer's actions into its developing policy, how the trainer's expertise influences the learner's ability to acquire an appropriate policy, and when the trainer should provide information to the learner. The results of our investigation will provide more information about integrated systems, with the aim of making them easier to develop and apply.

Based on the desire to answer these questions, we have designed, developed, and implemented an integrated system, calling it the ASK FOR HELP (AFH) approach. In AFH, the learner interacts with an automated trainer. To develop its policy, the learning agent relies on $Q$-learning (Watkins, 1989), a particular reinforcement learning algorithm. With reinforcement learning as its main learning component, AFH maintains many of the advantages of reinforcement learning, such as learning from the scalar rewards and developing optimal policies. In addition to depending upon the rewards for training information, the agent also acquires on-line, trainer-suggested actions as the learner performs the task. One of the key components of AFH is the strategy with which the learner acquires information from the trainer: the learner asks the trainer for help. Via two particular asking strategies, one of which is a simple stochastic approach, we can determine how much the trainer's actions help the learning agent acquire an appropriate policy. To assimilate the automated trainer's guidance, the learner simply executes the actions as though it had chosen them itself, thus incorporating them and their consequences seamlessly into its experience via $Q$-learning. We hypothesize that ASK FOR HELP is an effective approach to integrating apprentice learning and reinforcement learning. But, more importantly, AFH is a tool with which we can answer certain questions about hybrids of the two learning methods.

## 1.5 Overview of the Results

In this dissertation we examine the advantages of integrating apprentice learning and reinforcement learning. We utilize the ASK FOR HELP approach to answer certain fundamental questions about integrated systems, and demonstrate that AFH is an effective means for integrating apprentice learning and reinforcement learning. Our empirical study establishes that the learning agent can develop its policy more quickly

with AFH than with either reinforcement learning alone or apprentice learning alone, even with suboptimal trainers.

Generally, the more often the trainer provides an action to the learner, the more quickly the learner learns to perform the task. The learner requires the most time to develop a policy when it is not asking the trainer for aid, which is the same as learning with standard reinforcement learning. As the amount of aid increases, the speed with which the learner learns the task also increases; thus, the integrated approach is better than reinforcement learning alone. As the trainer begins providing a higher proportion of the training information to the learner—which approaches learning via apprentice learning—the learner sometimes learns more quickly when it can still perform its own actions occasionally, indicating that the new approach is better than apprentice learning. These results demonstrate that the integrated approach is better than either of its constituents alone, especially when the trainer provides suboptimal training information. Based on these results, one can make the tradeoff between the expense of employing a trainer and the expense incurred in terms of the time it takes to learn a policy.

The expertise of the training agent has a direct relationship on how quickly the learning agent finds an optimal policy. Trainers that provide optimal actions to the learner are the most helpful, and as the trainer's expertise degrades, the speed with which the learner finds an optimal policy also degrades. Although this result is not too surprising, we find that there is not much difference between learning from a trainer that provides optimal actions and one that gives an incorrect action a quarter of the time. This suggests that one does not need to have an optimal trainer in order to benefit by using an integrated approach. Thus, a human trainer, who may not be optimal, can be beneficial to the learning agent. Furthermore, this shows that the learning agent can learn to perform the task better than its trainer.

The speed with which the learner learns is not only influenced by the trainer's expertise, the timing of the trainer's actions is also important. The results of experiments in which the learner asks the trainer for aid randomly demonstrate that not all trainer's actions are as helpful to the learning agent as others. The same amount of interaction with the trainer—that is, the same number of trainer-suggested actions—can cause the learner to take widely varying amounts of time to find an optimal policy. This suggests that care must be taken in asking the trainer for help. Even so, it appears that a certain rate of interaction with the trainer leads most quickly to acquiring an optimal policy across a variety of problem sizes and trainer proficiencies. When the learner employs a more sophisticated strategy of asking for help, it can find an optimal policy more quickly with fewer trainer's actions. Thus, if the trainer were human, one would employ the more sophisticated asking strategy because it requires the least interaction with a human trainer.

From the empirical results, we conclude that one can indeed integrate apprentice learning and reinforcement learning to advantage. We demonstrate that our hybrid approach, in which the learner asks for aid randomly and simply executes the trainer's actions, allows the learning agent to produce correct policies more quickly than with either reinforcement learning alone or apprentice learning alone. We have also implemented a more sophisticated asking strategy that performs better than asking for aid randomly. Finally, the results show that the expertise of the trainer plays a major role in the success of the learner.

## 1.6 Guide to the Dissertation

The next chapter presents the two individual learning methods, apprentice learning and reinforcement learning, in greater detail. We review the work on these two methods and motivate our approach of integrating the two by exploring their disparate objectives, algorithms, and assumptions; comparing and contrasting their var-

ious characteristics. In Chapter 3 we discuss the issues that must be considered in integrating the two methods, and present a model of integration that not only provides a context in which to discuss related work and serves as a framework in which to explore the issues, but also describes how automated learners and trainers might interact. We begin Chapter 4 by posing questions that motivate our empirical study. Then, we present our instantiation of the hybrid method, the ASK FOR HELP approach (AFH). In Chapter 5 we begin by describing the particular domains on which we test AFH. We then present empirical results from a variety of experiments in which we attempt to find answers to fundamental questions about integrated learning. Finally, Chapter 6 summarizes the dissertation and presents issues for future research. We conclude with a discussion of our contributions.

# CHAPTER 2

# APPRENTICE LEARNING AND REINFORCEMENT LEARNING

This dissertation focuses on intelligent, computerized agents that learn to perform multiple-step tasks. The objective of the agents is to improve their policies so that they become more proficient problem-solvers. Two learning methods that attempt to endow the automated agent with the capacity to learn are apprentice learning and reinforcement learning. Each of these methods takes distinct approaches, with different assumptions, algorithms, and objectives.

In apprentice learning, the autonomous agent learns from examples of another agent's problem-solving behavior; and, in reinforcement learning, the agent's policy is derived based on scalar rewards received while performing the task. In his pioneering research in machine learning, Arthur Samuel employed approaches from each method to develop checkers-playing agents (Samuel, 1963; Samuel, 1967). Regardless of the particular learning method, each of his agents learned to play checkers moderately well, but could not beat master-level players. Since Samuel's time, much research has gone into developing apprentice learning and reinforcement learning further.

Before beginning a presentation of the two learning methods, we present the models of multiple-step tasks they each assume. Then, we present the learning methods in turn. Finally, we compare the methods along particular dimensions, such as the informativeness of the training information and the optimality of the learned policy. We show that each method has beneficial qualities, but that neither is clearly better than the other. We continue this line of reasoning in Chapter 3 by arguing that one

may want a hybrid learning method that has each of the desired qualities, and that strengths from each method may mitigate weaknesses in the other.

## 2.1 Models of Multiple-Step Tasks

Apprentice learning models multiple-step tasks as *state-space* problems. Reinforcement learning assumes the *Markovian decision task* model, from the fields of operations research and optimal control. These models abstract the details of the environment in which the task exists into a mathematical representation.

The main objective of this section is to point out that the two models are quite similar, differing only on a few details. Because the models are alike, considering a hybrid of the two learning methods is feasible. However, the disparities in the models give rise to some of the differences between apprentice learning and reinforcement learning, which we discuss below in Section 2.4.

### 2.1.1 State-Space Model

The state-space model, which is used frequently for studying problem-solving issues in Artificial Intelligence (Rich & Knight, 1991; Ginsberg, 1993; Luger & Stubblefield, 1993; Russell & Norvig, 1995), represents problems in terms of *states* and *operators*. The states are the situations that arise in solving the problem, such as the configuration of pieces on the chess board, and the operators, such as the legal piece moves, transform the problem from state to state, where the resulting state is called the *successor* state. Note that an operator is a function that maps states to states: applying an operator to a particular state produces another state. A state-space problem can be specified completely by an initial state, a set of operators, and a predicate that recognizes goal states. The objective in solving the problem is to find a sequence of operators—a solution path—that successively transforms the initial state into a goal state.

Figure 2.1 depicts part of a state-space problem, including a partial solution. The circles in the figure represent states, and the arrows depict operators, which transform the state at the arrow's tail into the state at the arrow's head. For example, the application of operator "a4" transforms "State A" into "State B." The dashed circles and arrows show the states not visited and the operators not taken in the solution.

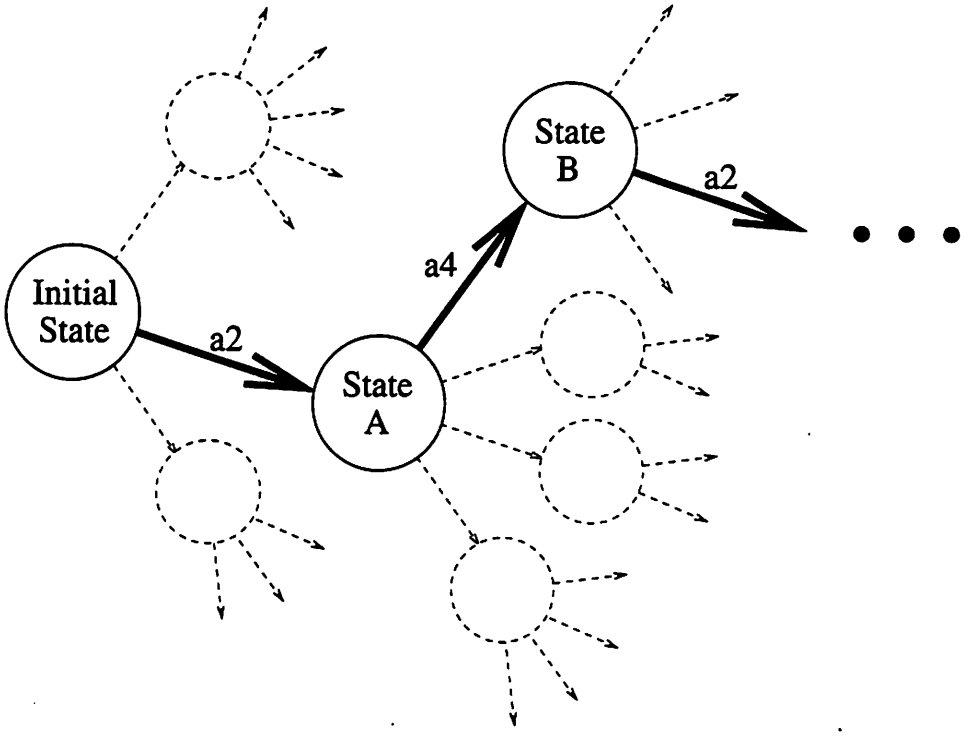



**Figure 2.1** State-space representation

In more complex state-space problems, real-valued weights are associated with the operators. Consequently, the model can take into account such concepts as the physical distance between a state and its successor (given that the states represent physical locations) or the actual cost of selecting the operator. In problems specified in this manner, the agent must optimize a measure of the weights; for example, finding the shortest, or least-cost, solution path. Accordingly, it is not sufficient to find any path to a goal.

### 2.1.2 Markovian Decision Task Model

The Markovian decision task (MDT) model was developed for studying stochastic control processes in operations research and control theory (Bellman, 1957; Puterman, 1994). Similarly to the state-space model, this model defines problems in terms of states, operators (called *actions*), and weights (called *rewards*)[1]. MDT's differ slightly from state-space problems on three particular points.

First, there is no goal recognition predicate. The sole objective in a MDT is to find a path through the space of states that optimizes a measure of the rewards received in performing the task. Incidentally, this does not restrict MDT's from modeling problems that require the agent to reach a goal state, because all transitions into the goal state can be given high reward.

Second, state transitions, the mapping of state to state by the actions, are stochastic. That is, the application of an action to a state may result in any one of a set of states, probabilistically. The state-space model, on the other hand, says little about the stochastic nature of state transitions, and most problem-solving systems assume the operators are deterministic.

Finally, MDT's are explicit about requiring that all future state transitions be conditionally independent of any previous states or action choices given the current state. This restriction, called the Markov property, makes MDT's amenable to rigorous mathematical study, but does limit the applicability of the model somewhat. Nevertheless, many interesting problems, such as robot control, can be modeled as MDT's. Indeed, the state-space model assumes the Markov property implicitly, and thus many problems studied in Artificial Intelligence have this property.

[1] In the sequel, we will use the terms 'action' and 'operator' interchangeably. This is also true of 'weights' and 'rewards'. We will consider 'solving a problem' and 'performing a task' to be synonyms for the activity of making a sequence of decisions to achieve an end.

## 2.2 Apprentice Learning

The learning agent in apprentice learning observes a training agent that is performing a multiple-step task. Figure 2.2 presents a diagram of the apprentice learning scenario, with a *task*, a *learner*, and a *trainer*. The trainer, which may be a human or an automated agent, selects the *actions* that are applied to the task and observed by the learner. Both the learner and trainer observe the *state* of the task.

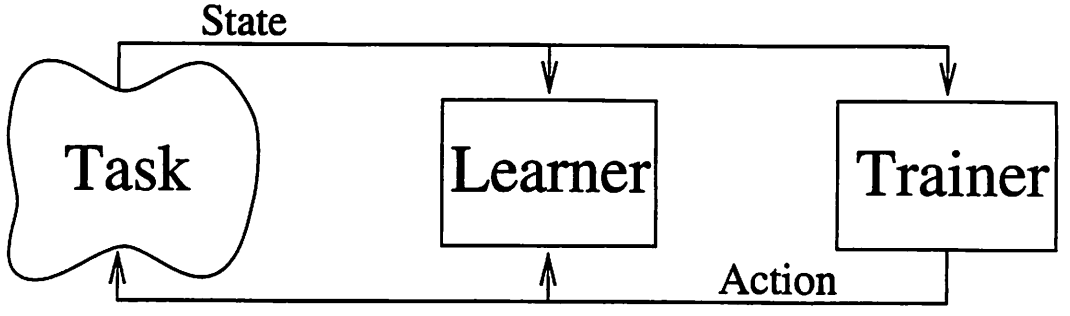



**Figure 2.2** Apprentice Learning Method

The objective of the learner is to develop a good policy $\pi$ that can be applied to the task. The agent attempts to meet this goal by learning to mimic the trainer. Because the trainer can perform the task, the assumption is that attempting to copy the trainer's strategy will allow the learner to do so also. At each discrete time step in the task, the trainer selects an action $a_t$ based on the current state $s_t$ and whatever knowledge the trainer may have about the task. The trainer then applies the action, transforming the task to the state $s_{t+1}$. Meanwhile, the learner has acquired knowledge about how to perform the task by noting the action that the trainer chose, acquiring a state-action pair $(s_t, a_t)$ that serves as an example of a correct decision.

This pairing of state with action is powerful training information: the learner has evidence that a particular action is applicable to a particular state. As the task progresses, the agent acquires a set of $(s_t, a_t)$ training examples that are representative

of each of the trainer's decisions along the entire solution path. Given the set of $(s_t, a_t)$ pairs, the learner can build its policy with any of the supervised learning techniques, such as decision trees (Quinlan, 1993; Utgoff, 1994) or neural networks (Nilsson, 1965; Rumelhart & McClelland, 1986). This type of learning has also been called "copying an existing controller" (Barto, 1990) and was used as early as 1964 for this purpose (Widrow & Smith, 1964).

### 2.2.1 Apprentice Learning Systems

Several systems have been developed that employ apprentice learning to develop a policy. In this section we describe three of them. The first is one of Samuel's checkers players (Samuel, 1967). More recently the method has been applied to training an automated agent to drive a van (Pomerleau, 1991) and to fly a simulated aircraft (Sammut, Hurst, Kedzier & Michie, 1992).

One of the earliest uses of apprentice learning was in training an agent to play the game of checkers (Samuel, 1967). The trainers are master checkers players, whose move choices from many games are recorded as book moves. The training games consist mostly of tied games, in which the entire sequence of moves made by each player is available to the learner. For games that are won by one of the players, only that player's moves are placed in the book. Note that humans also study book move information to improve their skills.

In learning to play, the automated agent simulates the re-playing of the recorded games, examining each game state in turn and adjusting its policy according to the move chosen by the master player. Samuel explores two types of policy, both of which pick the current move choice according to an evaluation function coupled with limited-depth search. For a given state, the agent changes the evaluation function so that it will increase the likelihood that the trainer's move will be chosen in the future by giving more worth to the trainer's recorded move and decreasing the value

of making the move choices that were not chosen. The automated agent learns to play checkers moderately well, but can not beat master checkers players.

In the Autonomous Land Vehicle in a Neural Network (ALVINN) project (Pomerleau, 1991), an automated agent learns to steer a modified van by observing a human's driving behavior. While the human drives, ALVINN creates a training example from the current state, represented by a video image of the scene ahead of the van, and the human's steering direction. ALVINN's policy is implemented as a multi-layer, feed-forward network (Rumelhart & McClelland, 1986) that takes the video image as input and determines the steering direction according to the values of its output units. Presented with a training example, the network adjusts its weights in an attempt to produce the requisite steering direction.

Observing the human does not provide enough training examples for ALVINN to learn to steer the van. Pomerleau notes that the learning agent is not supplied with enough variety of experience by simply watching the human. It cannot recover from its own mistakes nor variations due to the uncertainties inherent in steering a van. To remedy this situation, in addition to learning from each training example derived directly from the action of the human driver, ALVINN also learns from several simulated training examples that are based on perturbations of the video image and the corresponding change in the human's steering direction. With these additional examples, the agent learns to drive the slowly moving van successfully .

In the final apprentice learning system that we describe, an automated agent learns to fly a simulated aircraft by observing the actions of a human pilot (Sammut, et al. 1992). As the human flies the aircraft on a predetermined flight plan, the state information and the human's actions are observed by the learner and the training examples are recorded into a file. After the pilot completes the flight plan by landing the plane safely, the training information captured in the file is separated into seven training files, one for each stage of the flight (*e.g.* taking off, straight-and-level flight,

landing). The files are then processed individually by the inductive decision-tree learning algorithm C4.5 (Quinlan, 1993) to produce a policy, in the form of a set of rules, for every flight stage. Using this technique, autopilots are developed based on examples derived from three separate human pilots. Each of the three automated pilots is able to fly the pre-specified flight plan in approximately the same manner as the human from whom it had learned.

### 2.2.2 Apprentice-like Learning Methods

A different type of apprentice learning is represented by systems like LEAP (Mitchell, Mahadevan & Steinberg, 1985) and PROTOS (Bareiss & Porter, 1987). LEAP is a classification system for circuit designs. Whenever LEAP produces an incorrect circuit design for a specified requirement (*i.e.* misclassifies the requirement), the human trainer provides LEAP with the correct circuit design. PROTOS, is a classification system for objects, such as dogs and tables. PROTOS asks the human for help when it misclassifies an object. The human and PROTOS then engage in a dialogue that results in changes to PROTOS's knowledge about the classification task. Although both of these systems learn from a human training agent, and can therefore be classified as apprentice-like methods, neither attempts to learn a policy for problem-solving.

## 2.3 Reinforcement Learning

The learning agent in reinforcement learning updates its policy based on its own problem-solving experience. Figure 2.3 shows the major components of the reinforcement learning scenario: a *task* and a *learner*. These two components interact via *states*, which are the task situations; *actions*, by which the agent manipulates the

task; and *rewards*, the training information received by the agent after performing each action[2].

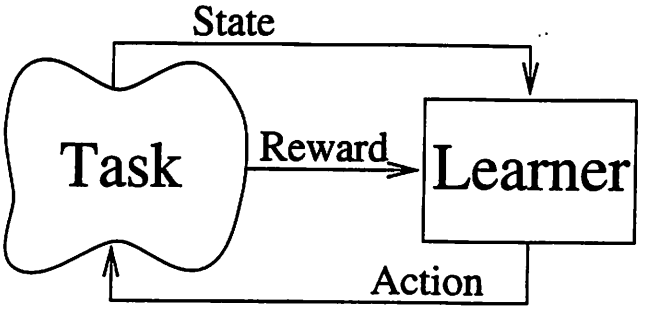



**Figure 2.3** Reinforcement Learning Method

At each time step in learning to perform the task, the learner selects an action $a_t$ to perform based on the current state $s_t$ and its developing policy $\pi$. After the agent applies the action, which changes the state of the task, it receives a reward $r_t$ (which may be zero) that indicates how well it is performing. This evaluative feedback is only weakly informative, simply revealing the short-term performance level of the agent and not giving specific information about the applicability of any of the previous actions. Nevertheless, the scalar rewards are sufficient for the agent to develop a policy that optimizes an additive measure of those rewards. In particular, the agent can learn to maximize the expected return, which is a weighted sum of the rewards received over the course of performing the task starting at a particular state. Stated mathematically, the agent attempts to maximize:

$$E\left(\sum_{t=0}^{\infty}\gamma^t r_t\right) \tag{2.1}$$

[2]The actual source of the rewards is generally considered to be the *environmental critic*, a component of the environment in which the task is embedded. The critic is abstracted away when the task is represented as a Markovian decision task: the rewards become simply labels on state transitions. Thus, we can treat the task itself as the source of the rewards. However, this distinction is not important here. What is important is that the learner receives reward signals, regardless of their actual source.

where $r_t$ is the reward received at time $t$, $0 \leq \gamma < 1$ is the discount factor, and $E$ signifies expected value (which is necessary because state transitions may be stochastic). Because of the discount factor $\gamma$, immediate rewards are weighted more heavily than rewards received in the future. The weighting depends on the value of $\gamma$. With values near 1, current and future reward are weighted almost equally, and lower settings place the emphasis on rewards received in the near future. Other cumulative measures of reward, such as the average reward (Schwartz, 1993), have also been studied, but the discounted case lends itself more easily to rigorous mathematical study, and is currently the most popular form of return considered in other research.

Because the learning agent is solely responsible for its own training experience it is faced with two inherent challenges. The first is the problem of temporal credit assignment (Minsky, 1963): Which of the many actions taken by the learning agent in the past should be credited, or blamed, for the most recently received reward? This issue is particularly difficult when the receipt of non-zero rewards is sparse. The other difficulty, called the explore/exploit tradeoff, arises because the agent must make the trade between performing well on the task now versus experimenting with its options in order to learn more about the task. By exploring, the learner may be able to develop a better policy than its current one. So, the learning agent has to decide when to take the action that its policy suggests versus when to choose another action that may give it more information about the task. Much progress has been made in solving the temporal credit assignment problem, and although formally justified exploration techniques have been developed, many current systems still employ ad hoc solutions to the explore/exploit tradeoff.

The remainder of this section provides more detail on three prevalent reinforcement learning approaches. See Kaelbling, Littman and Moore (1996) for an in-depth survey of reinforcement learning.

### 2.3.1 Temporal Difference Learning

The roots of temporal difference learning lie in one of the earliest reinforcement learning systems, the first of Samuel's checkers players (Samuel, 1963). The learner receives sparse rewards, given at the end of the game, that indicate only whether the game was a win, loss, or draw for the learner. In addition to these signals, the learner updates its policy—an evaluation function coupled with limited-depth search—according to its own assessment of how well it is performing. The agent adjusts its evaluation function so that the estimated value of the current state will move closer to the value of the best choice as determined by the limited-depth search. That is, the current state's evaluation changes to reflect the backed-up value of the highest-valued successor state. With this simple technique of backing-up state values, the agent learns to play checkers well enough to be considered a moderate checkers player. Note that there is no formal exploration policy: exploration results only from the changing evaluation function.

This idea of regressing values led to the development of *temporal difference learning* (TD) (Sutton, 1988), which is the main technique for dealing with temporal credit assignment in reinforcement learning. Via TD, the learning agent learns to predict the expected return $V(s)$ of each state,

$$V(s) = E\left(\sum_{t=k}^{\infty} \gamma^t r_t | (s_k = s)\right) \tag{2.2}$$

which is the expected sum of discounted rewards to be received if the agent is in state $s_k$ at time $k$ and uses the optimal policy thereafter. Temporal difference learning receives its name because the change to the predicted return is based on the difference between the previous prediction of the state's return $V(s_{t-1})$—the value of the previous state—and the current prediction $r_{t-1} + \gamma V(s_t)$—the sum of the actual reward received and the value of the current state. The current prediction is a more accurate measure of the actual value because it considers the reward received. After the agent

has executed action $a_{t-1}$ in state $s_{t-1}$, receiving reward $r_{t-1}$ and arriving at state $s_t$. the update for $V(s_{t-1})$ is:

$$\Delta V(s_{t-1}) = \alpha(r_{t-1} + \gamma V(s_t) - V(s_{t-1})) \tag{2.3}$$

where $V(s_{t-1})$ and $V(s_t)$ are the predicted returns of $s_{t-1}$ and $s_t$, and $r_{t-1}$ is the actual reward received. $\gamma$ is the factor that discounts rewards received in the future (see Equation 2.1), and $0 < \alpha < 1$ is the *step size* parameter that controls how much $V(s_{t-1})$ actually changes.

There is an entire family of TD algorithms, called TD($\lambda$), where $0 \leq \lambda \leq 1$ is a parameter that controls how much of the current difference $(r_{t-1} + \gamma V(s_t) - V(s_{t-1}))$ is applied to all previous states $s_j, j \leq t-1$. Equation 2.3 is the TD(0) update rule, in which only the most recent state's value is affected. For any particular state $s_j$, the amount of change to that state's value $V(s_j)$ is determined by the current difference and by the state's *eligibility* $e(s_j)$, which is a function of $\lambda$ that indicates how far in the past and how frequently the state has been visited. Equation 2.4 gives the TD($\lambda$) rule, which is applied at time $t$ to all states, $s_j, j \leq t-1$.

$$\Delta V(s_j) = \alpha(r_{t-1} + \gamma V(s_t) - V(s_{t-1}))e(s_j) \tag{2.4}$$

The eligibility of a state is straightforward to calculate. At each time step, each eligibility is updated as follows:

$$e(s_j) \leftarrow \begin{cases} \lambda e(s_j) + 1 & \text{when } s_j = s_{t-1} \\ \lambda e(s_j) & \text{otherwise} \end{cases} \tag{2.5}$$

Thus, the eligibility of a state is an exponentially decaying trace of when and how frequently the state was visited. The eligibility mechanism implements a form of credit assignment, because the values of previous states are affected by the current difference, to an amount determined by how far in the past the state was visited. The most recent states are affected most, and states in the distant past are affected very

little. Sutton (1988) and others (Dayan, 1992; Dayan & Sejnowski, 1994) note that TD($\lambda$) converges more quickly for $\lambda \neq 0$.

It has been proved that TD($\lambda$), under certain conditions, will converge to the optimal value function (Sutton, 1988; Dayan, 1992; Dayan & Sejnowski, 1994; Jaakola, Jordan & Singh, 1994). Via temporal difference learning, TD-Gammon (Tesauro, 1995), a backgammon-playing program, not only became the world's best computer backgammon player, but is also considered to be as good as the world's best human backgammon players.

### 2.3.2 Actor-Critic Architecture

The actor-critic architecture (Barto, Sutton & Anderson, 1983) consists of two components: the policy—also called the Associative Search Element (ASE)—and the evaluation function—known as the Adaptive Critic Element (ACE). This algorithm is closely related to the method of policy iteration in dynamic programming (Bellman, 1957; Barto, Bradtke & Singh, 1995), which is a classical, exhaustive method for finding optimal policies.

The ASE and ACE are each faced with different learning problems. In its role as evaluation function, the ACE maps states to real values, and attempts to predict the value $V(s)$ for each state (Equation 2.2). The ACE's update rule is the TD(0) rule given in Equation 2.3, which changes the previous assessment of predicted reward based on the current evaluation and the actual reward received.

As the policy, the ASE maps states to actions. Each time the learner performs an action, it updates the ASE according to the immediate (predicted) reward from the ACE. Because the ASE receives immediate feedback, it does not have to deal with the credit assignment problem. However, since the ACE is being developed simultaneously with the ASE, there is much uncertainty in the training signal used by the ASE early in training. Indeed, there is no proof that the ASE will converge to an optimal policy.

The ASE/ACE pair was first applied to learning to control the cart-pole task (Barto, Sutton & Anderson, 1983). Controlling the cart-pole involves keeping a pole balanced on a movable cart and keeping the cart within the boundaries of a fixed-length track by applying a bang-bang control policy. In the system built by Barto, et al. (1983), the ASE/ACE were each implemented as a linear threshold unit (Nilsson, 1965). In later work, Anderson (1989) built each with a multi-layer feed-forward network. Both of these implementations were able to learn to control the cart-pole task after extensive experience.

### 2.3.3 $Q$-learning

Instead of maintaining both a policy and an evaluation function, $Q$-learning (Watkins, 1989) combines them into a single mechanism. This reinforcement learning method is closely related to the classical dynamic programming method of value iteration (Bellman, 1957; Barto, et al. 1995), in which an evaluation function of the states is developed and the derived policy is greedy with respect to that function.

For each state-action pair $(s, a)$, $Q$-learning estimates the return of the state $s$, given that the action $a$ is performed and the optimal policy is followed thereafter. The quantity $Q(s, a)$ is called the $Q$-value of the state-action pair. Given a state $s$ and the set of actions $A$ that can be executed at that state, the implied policy selects the action that has the highest $Q$-value. That is:

$$\pi(s_t) = \arg\max_{a \in A} Q(s_t, a) \tag{2.6}$$

The $Q$-function is trained similarly to the ACE discussed above, with temporal difference learning. In particular, the previous assessment of return, $Q(s_{t-1}, a_{t-1})$ is changed to reflect the currently predicted evaluation, $r_{t-1} + \gamma \max_{a \in A} Q(s_t, a)$. Remember that the optimal action to perform in state $s_t$ is the action that produces the

highest $Q$-value (see Equation 2.6), and so $\max_{a \in A} Q(s_t, a)$ represents the predicted value of taking that action. The update rule is given by:

$$\Delta Q(s_{t-1}, a_{t-1}) = \alpha(r_{t-1} + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})) \tag{2.7}$$

This update can also be expressed in a form similar to TD($\lambda$) where the current difference also applies to updates on previous states.

It has been proved that $Q$-learning under certain restrictive conditions, will converge in the limit to the optimal evaluation function (Watkins, 1989; Watkins & Dayan, 1992; Jaakola, Jordan & Singh, 1994; Tsitsiklis, 1994). These proofs, although they do not say much about how $Q$-learning will perform in practice, lend credence to the widespread use of $Q$-learning. Indeed, several researchers have employed $Q$-learning with success: in robotics and maze tasks (Mahadevan & Connell, 1992; Lin, 1992; Singh, 1992) and in elevator dispatching (Crites & Barto, 1996).

### 2.3.4 Model-Based Reinforcement Learning

Each of the reinforcement learning methods discussed so far are examples of model-free algorithms. Although the learners in model-free algorithms are exposed to the transition probabilities and the reinforcements received for each state-action pair experienced, they do not try to learn a model of either of those aspects of the task. The model-based reinforcement learning algorithms, such as Dyna (Sutton, 1990; Sutton, 1991), Prioritized Sweeping (Moore & Atkeson, 1993) and the closely related Queue-Dyna (Peng & Williams, 1993), do learn such models. Because they have a model of the task, they can rely on it in order to simulate the outcome of the application of actions to arbitrary states; thus, gathering more experience in the task than they can on-line. These approaches can be costly in terms of computation. In addition, the learned policy is based on the imperfect model. Nevertheless, the model-based algorithms typically perform better than the model-free algorithms.

## 2.4 Comparison

Apprentice learning and reinforcement learning both endow the automated agent with the capacity to learn to solve problems, allowing the learner to develop as good a policy as it can. In achieving this end, they each take differing approaches, with disparate assumptions, objectives, and algorithms. One can characterize the methods along dimensions such as: the *availability* of training information, the *informativeness* of the training information, the amount of *exposure* the learner has to the state-space, and the *optimality* of the learned policy. One can also discuss how each method deals with *stochastic* and *temporal* aspects of the task, and with the *credit assignment* and *explore/exploit* problems. Finally, another quality on which the methods differ is the *speed* with which each achieves its objective. One can compare and contrast the two methods along these dimensions and notice that neither one is clearly preferred over the other.

When setting up a problem to be learned, one must be concerned with the availability of training information. Without this information, the task cannot be learned. In apprentice learning, the source of the training information is a human (or other agent) that can perform the task completely, but not necessarily optimally. For any given task, finding a sufficiently skilled trainer may not be possible, thus limiting the applicability of the method. Reinforcement learning requires only that the agent receive evaluative rewards in the course of problem-solving. The advantage of relying on rewards for training is that many problems can be cast easily as reinforcement learning problems, in the extreme associating appropriate rewards with only successful or failed end states.

Because apprentice learning and reinforcement learning each have disparate sources of training information, one may ask how informative is each type of information. A strength of apprentice learning is that the learner receives prescriptive training. Because a human that knows how to perform the task is the source of the state-action

training pairs, it can be assumed that the information is useful for developing a good policy. For example, as Samuel points out, book moves are "representative of the very best play" (Samuel, 1967, p. 612). Although the trainer's choices may not always be optimal, they do give a strong indication of how to solve the problem. Moreover, the agent learns from entire sequences of state-action pairs that are known to lead from the start state to the completion of the task. In reinforcement learning, the training information is not so informative. It does not reveal which action should have been chosen nor which of the many previous actions led to the receipt of the reward. Furthermore, non-zero reward signals may be infrequent, providing the agent with only sparse training information. A learning agent should have access to the most informative training information possible.

The two methods also differ in the amount of exposure to the task that the agent acquires. In apprentice learning, because the learner observes the solution path of the trainer, it only receives training within a very narrow portion of the state-space. Accordingly, the agent's performance may suffer if it were to stray from that narrow area. This situation is remedied in ALVINN by providing the agent with derived training information, representative of the state-space in a neighborhood of the solution path. In reinforcement learning, the agent must explore large portions of the state-space to gather information about which actions are the best to choose. Sammut et al., in their work in apprentice learning, observe that exposure to more of the state-space can lead to better policies. They note that the worst human pilots, because they made mistakes and then had to recover from them, exposed the automated agents to more of the state-space than just the narrow solution path, and were the source of the best autopilots. Thus, one desires that the learning agent gain action-selection experience over a wide range of states.

The optimality of the learned policy is also a characteristic of concern. There is no evidence that an agent employing apprentice learning will learn an optimal policy.

The objective of the learner is simply to learn a policy that is based on the trainer's actions. Empirical results do not say much about the "goodness" of the learned policy. The master checkers players, who were by definition proficient at checkers, did not help Samuel's checkers players learn to perform optimally. However, Sammut et al. found that two of the learning agents could perform the task better than their trainers, due to generalization. For apprentice learning, we barely have empirical evidence that the learner will develop a useful policy at all, let alone an optimal one. On the other hand, there is much theoretical support for the claim that a reinforcement learning agent will eventually learn an optimal policy. Many convergence proofs for TD($\lambda$) and $Q$-learning exist (Watkins, 1989; Watkins & Dayan, 1992; Dayan, 1992; Dayan & Sejnowski, 1994; Jaakola, Jordan & Singh, 1994; Tsitsiklis, 1994), mostly due to the links reinforcement learning has with dynamic programming and stochastic approximation. Clearly, having theoretical support for the convergence properties of the learning algorithm is a preferred quality.

Each of the methods approaches learning about the stochasticity and temporality of the task differently. Here, stochasticity refers to the fact that applying an action to a state may result in any one of a set of states, probabilistically. Temporality refers to the fact that an agent's current actions will impact the rewards it receives in the future, and will influence whether the agent is capable of completing the task. Neither of these issues are dealt with explicitly in apprentice learning. The agent simply learns to mimic the trainer's actions, under the assumption that the trainer knows the correct actions to perform. Reinforcement learning, though, handles both explicitly. The agent builds its policy to optimize the return, which is not only defined in terms of future rewards but is also an expectation over those rewards. Because of the expectation, the agent considers the stochastic nature of the task. Furthermore, the agent takes into account the rewards that may be received in the future, because

the measure of optimality and, hence, the returns that are to be estimated are defined in terms of the current and all future rewards.

Reinforcement learning must face the credit assignment problem and the explore/exploit tradeoff. Learners in apprentice learning need not concern themselves with these two issues. First, there is no credit to assign. When the learner receives a state-action pair, it can learn directly that the action should be evoked in that state. There is no reason for the apprentice learning agent to explore either: the agent just learns from the trainer's actions. Reinforcement learning, by its very nature, must be able to handle credit assignment and exploration. Having to deal with temporal credit assignment and exploration of the task makes learning more difficult.

Finally, the two methods reach their objectives at different rates. In apprentice learning, the agent attempts to build as good a policy as it can based on observation of the trainer. In all three apprentice learning systems described above, the agent is able to do this quickly: Running an induction algorithm on a set of $(s_t, a_t)$ training instances does not require a prohibitive amount of time. We must point out, though, that the agents do not learn to perform the tasks in the same manner as their trainers. Samuel's checkers player does not learn to play like a checkers master, and two of the autopilots learn to fly better than their trainers. Reinforcement learning methods generally require huge amounts of time to develop policies for even simple tasks. Remember, though, that reinforcement learning is attempting to build an optimal policy, not just a satisfycing one. Furthermore, reinforcement learning can be faster than the only other method that produces optimal policies, dynamic programming, and, unlike dynamic programming, does not require the complete specification of the task. To some extent, one cannot compare the two methods on this criterion of speed because the objectives are so different.

We have compared and contrasted the two learning methods on particular dimensions. The comparison is summarized in Table 2.1. Apprentice learning provides

the learner with higher quality training information, and does not have to be concerned about credit assignment nor the explore/exploit tradeoff. On the other hand, reinforcement learning exposes the trainer to a larger portion of the state space, guarantees optimal policies under certain conditions, and manages the stochastic and temporal aspects of the task explicitly. In our comparison of the two learning methods, we point out that on any given dimension one of the methods is preferred, but that neither method is preferred on all dimensions.

**Table 2.1** Comparison of Apprentice Learning to Reinforcement Learning on a select set of characteristics. The preferred quality is in bold-face type.

| Characteristic | Apprentice Learning | Reinforcement Learning |
|---|---|---|
| Training information available | rarely | **often** |
| Quality of training information | **prescriptive** | evaluative |
| Exposure to problem space | narrow | **broad** |
| Optimality of policy | rarely | **approaches optimal** |
| Stochasticity | implicit | **explicit** |
| Temporality | implicit | **explicit** |
| Credit assignment problem | **no** | yes |
| Explore/exploit problem | **no** | yes |
| Speed objective satisfied | **fast** | slow |

# CHAPTER 3

# INTEGRATING APPRENTICE LEARNING AND REINFORCEMENT LEARNING

Apprentice learning and reinforcement learning each give an automated agent the capacity to learn to perform multiple-step tasks. They both allow the learner to improve its policy and become more accomplished at the task. In our analysis of the two methods in the previous chapter, we describe particular dimensions of comparison and note that the two methods are clearly different. A learning method that possesses a higher proportion of the desirable characteristics will be considered an improvement over either of these methods. One can develop such a preferred method through a prudent integration of the two individuals. The new hybrid method will not only have many of the beneficial qualities of its constituents, but it will also reduce their weaknesses via the synergistic combination of its complimentary components.

Hybrids of apprentice learning and reinforcement already exist (Utgoff & Clouse, 1991; Clouse & Utgoff, 1992; Lin, 1992; Lin, 1993), and have been shown to perform better than their constituents. For example, Utgoff and Clouse (1991) developed an integrated learning agent that only needs to traverse the task once from the start state to goal state in order to learn to perform it correctly. Learning without the integration requires more training: two traversals of the task when learning via apprentice learning and almost five hundred traversals when learning via reinforcement learning. The learning agent in Clouse and Utgoff (1992) achieves learning speedups of up to two orders of magnitude versus reinforcement learning alone. Finally, Lin (1992, 1993) shows that giving a learning agent the ability to incorporate a trainer's knowledge improves the speed with which the agent learns. Each of these systems

hints at the positive benefits of integrating apprentice learning and reinforcement learning and serves as inspiration for our work.

## 3.1 An Integrated Learning Method

We define an integration of apprentice learning and reinforcement learning to be a learning method in which the learner can improve its policy based on rewards and on actions provided by a trainer. The trainer may or may not be human, and does not need to be expert at the problem. Figure 3.1 depicts such an integrated learning method. The relationship between the learner and the task is the same as that in reinforcement learning: at every time step, the learner observes the state, chooses an action to perform, and receives a reward. The relationship between the learner and trainer is similar to that in apprentice learning because the learner can observe actions that the trainer supplies. Unlike apprentice learning, the trainer does not need to provide an action to the learner at every step of the task. Either the learner or trainer determines when it is appropriate for the trainer to give an action. Accordingly, the interactions between the learner and trainer are more sophisticated, for example allowing the learner to ask the trainer to provide it with an action.

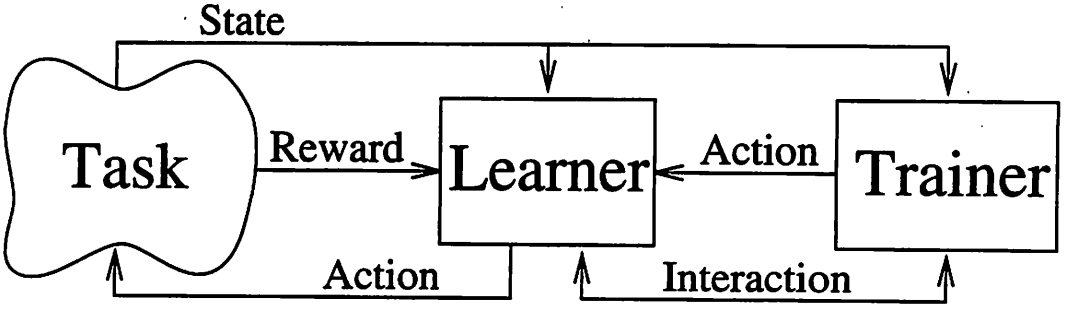



**Figure 3.1** Integrated Learning Method

According to our definition, the learner receives task knowledge from the trainer in the form of actions to perform. This approach has appeal for two main reasons. First, because apprentice learning relies solely on the actions provided by the trainer, it seems reasonable to do the same in a method that has apprentice learning as one of its components. More importantly, having the trainer give actions is a natural and easy way for the trainer to provide information to the learner. The trainer does not need to codify knowledge into rules, but simply reacts to the current situation, providing the action that is deemed best. Actions, though, are not the only form of information that the trainer may be able to provide (see Section 3.4). However, in our work we limit the trainer in the integrated method to providing simple actions to the learner.

### 3.1.1 Desirable Characteristics

An integration of apprentice learning and reinforcement learning should have the following characteristics:

1. It should use whatever training information is available, whether from the environment or from the trainer, and be able to incorporate that information into its policy.
2. It should expose the learner to enough of the state-space for the learner to develop an effective policy, but should not require the learner to gain experience across the entire state-space.
3. It should allow the learning agent to develop a policy that optimizes the long-term receipt of reward signals, thus handling the stochastic and temporal nature of the task explicitly.

From the reinforcement learning component, the learner has access to scalar reward signals; and through apprentice learning, the learner receives actions from the trainer. An integrated method must be able to assimilate each type of information into the policy. The difficulty here is in designing a learning algorithm that updates

the policy according to two disparate forms of knowledge. The algorithm must adjust the policy according to the scalar signals, and it must change the policy based on the trainer's actions.

The learner must also be exposed to broad areas of the state space. As Sammut et al. (1992) and Pomerleau (1991) describe, experience with a wide selection of state-action choices is important for learning a good policy. On the other hand, because it takes time to visit many states, the exposure should be focused on only those areas of state-space that are necessary to learn a policy that is optimal for the problem.

The ultimate goal of the learning agent is to develop an optimal policy. Finding such a policy is guaranteed (under certain restrictive conditions) by reinforcement learning. In an integrated method, using the trainer's information should not preclude the learner from developing an optimal policy, but should help it achieve such a policy more quickly. Being able to develop an optimal policy implies that the learning agent is taking the temporal and stochastic aspects of the task into consideration explicitly.

### 3.1.2 Issues in Integrating the Two Methods

Integrating apprentice learning and reinforcement learning raises a set of questions that we address in our research:

1. How should the learner and trainer interact?
2. How should the learner incorporate the trainer's proffered information into its developing policy?
3. When is it effective for the learner to receive training actions?
4. To what extent does the expertise of the trainer affect the learning?

Because the learner has access to reward signals and can update its policy according to those signals, it does not have to receive an entire sequence of state-action pairs from the trainer. In fact, the learner does not have to receive *any* information from the trainer at all, but should be able to use whatever information the trainer provides.

Because the learner has access to reward signals, the trainer is free to provide actions at unspecified times in the training. That is, the relationship between the learner and trainer is much richer than simply having the trainer give an action at every time step. To describe this more sophisticated relationship, we have developed a model of the interactions that take place between the learner and trainer. The model is inspired by the complex relationships between human learners and trainers. Even so, the model does not attempt to describe all possible interactions that may occur. It is described in Section 3.2.

One must also be concerned with the algorithmic detail of how to incorporate both reward signals and state-action pairs into the developing policy. It may seem surprising that one can produce a policy from both types of information, but several approaches have been proposed that work well (see Section 3.3.2). Unfortunately, it is unknown which is best and which gives the integrated method the desired qualities described above.

We also do not know when the trainer should interact with the learner to provide training information. Apprentice learning and reinforcement learning represent extremes of interaction; the trainer either provides information always or never, respectively. We explore the spectrum of interaction between these ends. The amount of trainer interaction becomes an important issue when one considers the cost of having the trainer in the scenario. One may like to have as little interaction with the trainer as possible, both to reduce the cost and to decrease the dependence of the learner on the trainer.

Finally, because the learner bases part of its policy on the trainer's actions, it seems natural to ask how good the trainer must be in order for the learner to benefit from those actions. In previous work (see Section 3.3.2) it is not clear how good the trainer is at performing the task nor how the trainer's expertise affects the learner's ability to develop an appropriate policy. Although we know that the trainer does

not have to be able to solve the problem perfectly from every state in order to be effective, we do not know how often the trainer can make mistakes.

### 3.1.3 Two Viewpoints

One can think of the integration of apprentice learning and reinforcement learning from at least two distinct viewpoints. First, one can imagine the hybrid as augmenting reinforcement learning with apprentice learning characteristics; for example, allowing the automated agent to learn also from a trainer's knowledge. One can also see the integration as adding reinforcement learning characteristics to apprentice learning. For example, in the hybrid method the learner can now deal explicitly with the temporal aspects of the task, when before it could not. Regardless of the particular viewpoint, each constituent brings positive characteristics to the hybrid. Furthermore, the characteristics of the individual methods interact, reducing each other's weaknesses.

In the integrated method, certain characteristics of apprentice learning will improve upon reinforcement learning's drawbacks. By having a trainer in the scenario, the learner has access to informative training information, which can augment the evaluative reward signals the learner already receives. As a different type of task knowledge, the trainer's actions will also aid the model-based reinforcement learning approaches. The trainer's actions may also help focus the efforts of the learner on parts of the space that are appropriate for learning an effective policy quickly, combating a major weakness of reinforcement learning described by Kaelbling et al.:

> One thing that keeps agents that know nothing from learning anything is that they have a hard time even finding the interesting parts of the space; they wander around at random, never getting near the goal. (1996, p. 275)

The trainer may help the learner find the "interesting parts" of space, showing the learner where it is profitable to explore and thus alleviating the explore/exploit difficulty.

In an integrated method, characteristics of reinforcement learning can reduce weaknesses of apprentice learning. One drawback of apprentice learning is that the trainer must be able to perform the entire task correctly. Because the learner in the hybrid method can now adapt based on reward signals, it does not need to rely as heavily on the trainer. In the hybrid method the trainer does not need to perform the task completely, from a start state to a goal state; instead, it may give partial solutions, some only one step long. The learner can also deal with the trainer's mistakes, refining incorrect training information in its continued exploration of the task. The addition of reinforcement learning characteristics to apprentice learning will also allow the learner to receive more exposure to the task than the narrow solution path of the trainer because exploration of the state-space is a key component of reinforcement learning. Moreover, reinforcement learning also brings its other strengths to the integration, dealing explicitly with the stochasticity and temporality of the task, and having a sound theoretical base.

## 3.2 A Model of Learner/Trainer Interaction

The relationship between the learner and trainer is more sophisticated in the integrated method than in either apprentice learning or reinforcement learning. The model of learner/trainer interaction introduced here specifies the interactions that may take place. The goal of the learner is to develop an optimal policy. The trainer shares this goal, providing the learner with actions so that the learner can improve. However, the model takes into account that the trainer may have its own work to perform and so must balance helping the learner versus completing its own task. The tasks of the learner and trainer are in the same domain, but may deal with different

aspects of that domain. For example, the learner may be learning how to navigate a robot in a fixed environment, whereas the trainer's task involves navigating around moving obstacles. A research issue that we explore is how much the trainer's expertise at the task influences how well the learner learns.

In the learner/trainer model, the learner and trainer employ two classes of actions. The first class of actions are those that transform task states, such as moving a robot forward, or turning the robot. The second class of actions, which we call meta-actions, define the interactions that occur between the learner and trainer. To distinguish between the mechanisms for deciding which action to choose and which meta-action to choose, we will continue referring to the first as simply a *policy* and will refer to the other as an *interaction policy.* A set of interaction policies are employed by the trainer and learner to determine which meta-action to choose. Although our work focuses on learning the policy, having the agents learn the interaction policies is a possibility for future research (see Section 6.2.3).

The following sections present more details on the learner/trainer model, describing the learner, the trainer, and the interactions between them.

### 3.2.1 The Learner

The learner is a computerized agent that is attempting to learn to perform a task by developing an optimal policy. Figure 3.2 depicts the model of the learner. The figure represents a simple conditional branch (diamond), the meta-actions (circles), and the interaction policies (stars). The conditional branch at the top of the diagram represents the fact that at each time step the learner determines first whether the trainer has provided it with any training information.

The learner makes use of three interaction policies, each of which pertains to different aspects of the interaction with the trainer. If the learner has not received a task action from the trainer, it can decide whether to ask the trainer for help. When the learner decides to ask for help, it performs the meta-action "Ask for Help" and

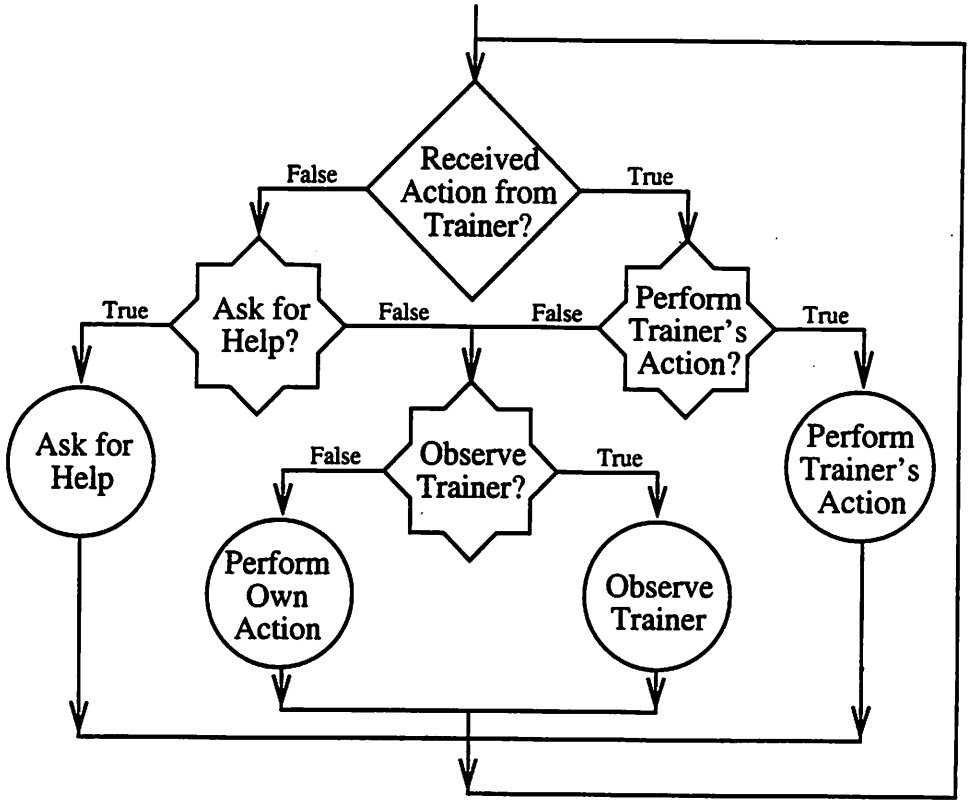



**Figure 3.2** Learner Model

sends a signal to the trainer notifying it of the request. The learner can decide to ask for help for many reasons; for example, when the it does not know what to do at that point. The learner may decide not to ask for help because it has determined that the trainer's help is not necessary. It is a research issue, which we explore herein, to determine when it is appropriate for the learner to ask for help.

When the learner receives training information from the trainer, it must decide the disposition of that information. The interaction policy "Perform Trainer's Action?" determines whether to learn from the trainer-supplied information. The learner may decide to use the trainer's information because the learner has just started learning. Or, the learner can decide to ignore the trainer's help, which it may do if it determines

that it has a better policy for the problem than the trainer or that the trainer is providing incorrect actions.

The final interaction policy, "Observe Trainer?," comes into play only when the learner has decided not to ask for help and not to learn from the trainer's proffered information. This policy determines whether the learner should simply observe the trainer in order to gain information about how to perform its task. In observing the trainer, the learner does not perform an action in its own task, but examines the trainer's task state and chosen action. The learner can then learn from that state-action pair. The learner may decide to do this, for example, when the learner's and trainer's tasks are similar to each other.

The final meta-action is executed when the learner has decided not to perform any of the other meta-actions. Then, the learner picks a task action to perform. As in reinforcement learning, the learner is faced with the problem of choosing an action that will exploit the information it has already gleaned about the task or performing an action to explore the task and gather more information.

### 3.2.2 The Trainer

The trainer may be an automated agent or a human and may be performing a task within the same domain as the learner. Figure 3.3 shows the model of the trainer. Like the figure of the learner model, this figure depicts a simple conditional branch, meta-actions, and interaction policies.

The conditional branch determines whether the learner has asked for help. If so, the trainer employs its interaction policy labeled "Give Solicited Action?" to determine whether to supply the learner with the requested action. The trainer may give the learner help because the learner has just started learning, or the trainer recognizes the learner's current task state as being difficult. The trainer may refuse to give help because it can not determine what action is appropriate to give because it does not have knowledge about that part of the space.

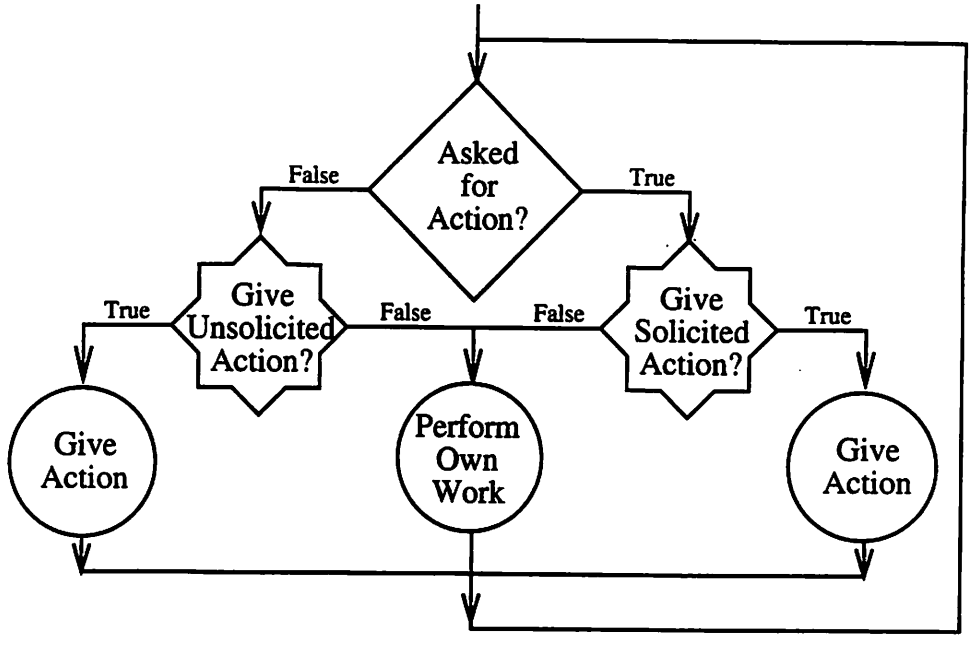



**Figure 3.3** Trainer Model

When the trainer decides to ignore the learner's plea for help, the trainer performs its own work (if it has any), deciding which action to apply to its own task. If, on the other hand, the learner's request is to be granted, the trainer provides the learner with the action that it determines to be appropriate for that state. The proffered action might not give the learner the optimal choice to make in that situation, because the trainer might not know the optimal action. The trainer is attempting to help the learner, though, and will not mislead the learner intentionally.

When the learner does not ask for help, the trainer decides whether to volunteer information, which it may do for the same reasons as offering solicited help. The meta-actions that can be applied by the "Give Unsolicited Action?" interaction decision-policy are the same as those under the interaction decision that pertains to the learner asking for help. Although the two interaction policies could be collapsed into only one, there may be a need for the trainer to maintain the distinction of whether the learner asked for help or not. Furthermore, trainers in currently existing

systems are either of one type, offering unsolicited help, or the other, offering solicited help. Thus, the model has two, separate interaction policies.

## 3.3 Fitting Previous Work to the Model

Several learning systems have been developed that allow an automated agent to learn to perform multiple-step tasks. Each has made assumptions about the interactions between the learner and trainer, employing particular mechanisms for allowing the learner to acquire task knowledge. In apprentice learning, which is at one end of the spectrum of interactions, the learner always acquires information from the trainer and always updates according to that information. At the other extreme is reinforcement learning, in which there is no trainer. Several researchers have also suggested systems that lie between these two extremes. The following sections describe these learning systems in more detail, discuss the assumptions about the interactions between the learner and trainer, and show how the methods fit the learner/trainer model.

### 3.3.1 Apprentice Learning and Reinforcement Learning Revisited

Both apprentice learning and reinforcement learning are specializations of the learner/trainer model, in which different aspects of the model have been left out. Figure 3.4 depicts a particular interpretation of apprentice learning. The shaded areas of the figure cover the parts of the model that are not applicable to apprentice learning. The learner, depicted on the left of the figure, can execute only one meta-action: perform the action provided by the trainer. The only applicable interaction policy "Perform Trainer's Action?" produces a fixed value of "True". At every time step, the learner receives an action from the trainer, and has no choice but to train with that action.

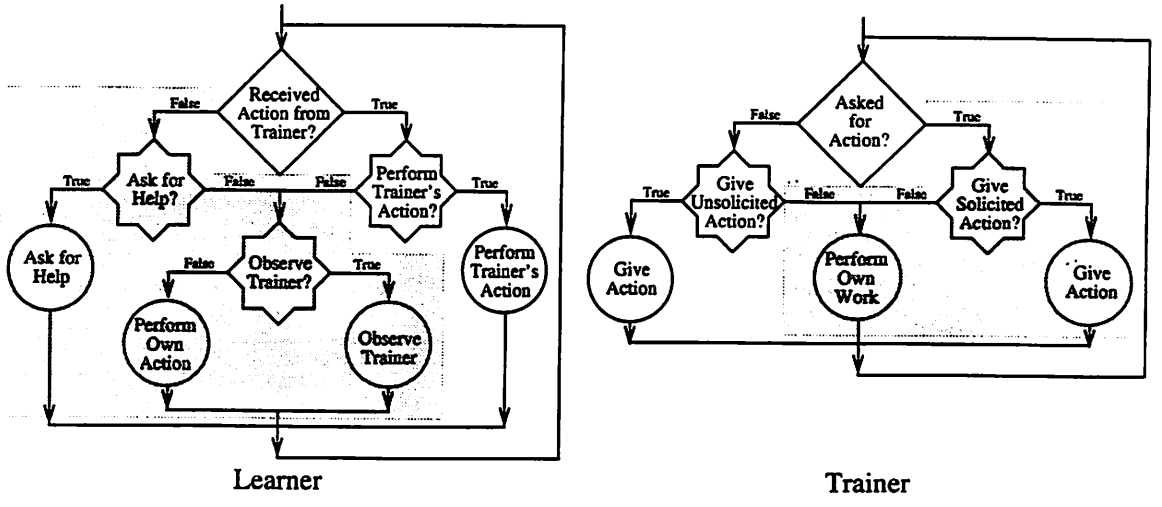



**Figure 3.4** Fitting Apprentice Learning to the Learner/Trainer Model, first interpretation

The trainer, depicted on the right of Figure 3.4, is never asked for help, but always provides it to the learner. At every time step, the trainer assesses the situation in the task and gives the learner the appropriate action. Like the learner, the trainer has only one meta-action to execute, "Give Action," and the interaction policy "Give Unsolicited Action" produces a fixed "True" value.

In a second interpretation of apprentice learning, the trainer performs its own work while the learner observes unobtrusively. This viewpoint also fits the proposed model, but different parts of it apply, as shown in Figure 3.5. The learner still executes only one meta-action, observing the trainer. Similarly, the trainer performs only one meta-action as it does its own work. The interactions between the learner and trainer are just as inflexible in this interpretation as in the other. The main difference between the two viewpoints is that in the first the learner is performing its own task with the help of the trainer, and in the latter the trainer is performing its own task with the learner observing. Both interpretations are appropriate and both exhibit the rigidity of the interactions in apprentice learning.

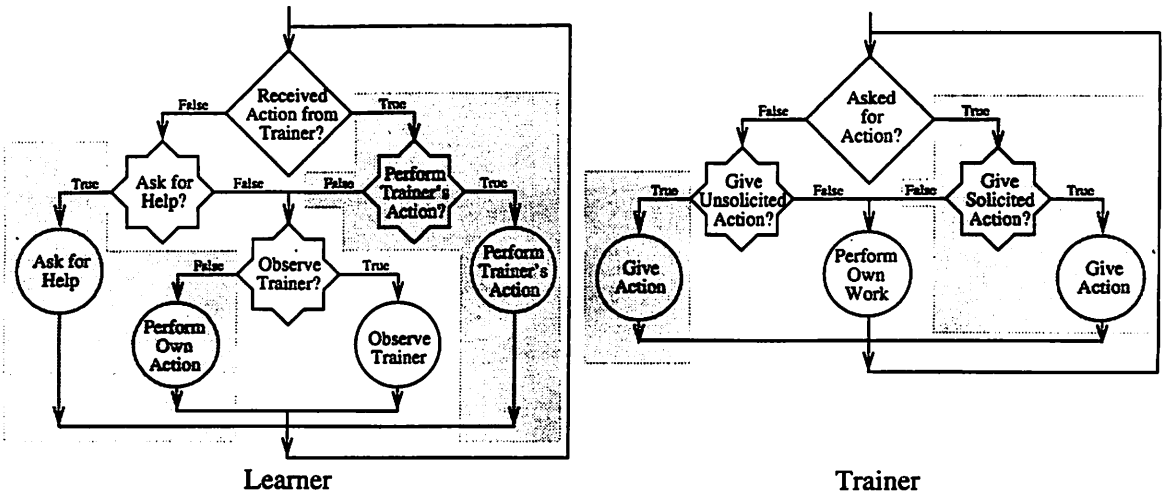



**Figure 3.5** Fitting Apprentice Learning to the Learner/Trainer Model, second interpretation

Figure 3.6 shows how reinforcement learning maps to the learner/trainer model. Via reinforcement learning, the learner relies solely on its own experiences to improve its performance. A trainer is not present to guide the learner, hence the model of the trainer, on the right of the figure, is completely shaded out. As shown in the left of the figure, both applicable interaction policies produce constant outputs. The learner can only perform its own actions, either doing the current best action or choosing an action for exploration purposes. And, the learner never receives help from a trainer because a trainer is not present. For the same reason, the learner cannot ask the trainer for help nor observe the trainer. The learner/trainer interactions do not exist in reinforcement learning.

### 3.3.2 Integrated Learning Methods

The integrated approaches can be divided into two main categories, based on the interactions they have with the trainer:

1. The trainer provides actions whenever the trainer feels it is necessary.
2. The trainer provides actions only when asked by the learner.

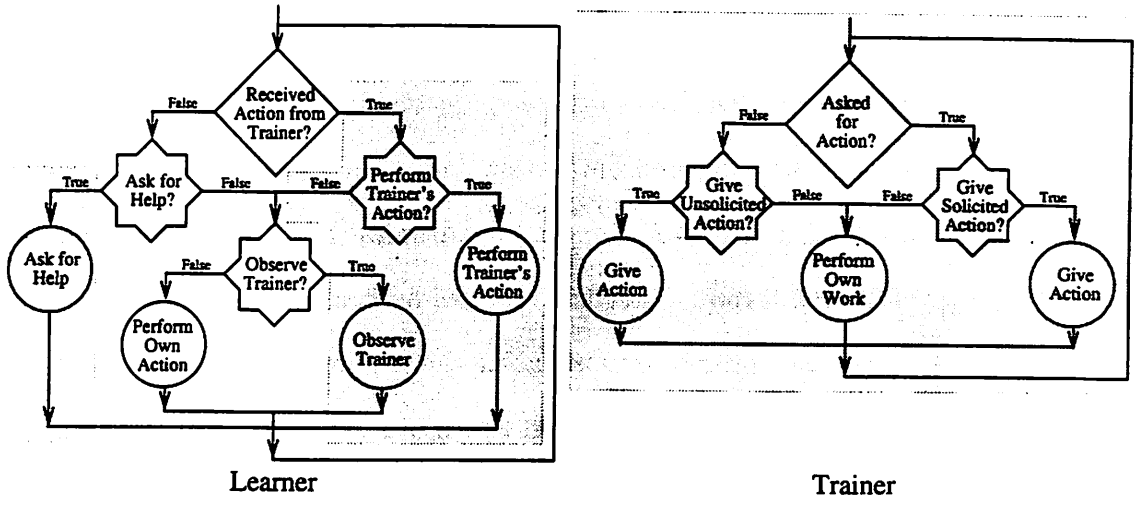



**Figure 3.6** Fitting Reinforcement Learning to the Learner/Trainer Model

In each of these categories, the learner always trains on the information provided by the trainer. The following two sections describe these systems and show how they fit the learner/trainer model.

Each of these systems hints at the power of integration. Experimentation with all of these systems shows marked decreases in the amount of training necessary for the learner to learn to perform the task. These performance improvements serve as an inspiration for our work of studying the integration of apprentice learning and reinforcement learning.

#### 3.3.2.1 Receiving Unsolicited Actions

The two systems presented here show that allowing a trainer to provide occasional, unsolicited information to the learner will improve the learner's ability to learn its task.

**Learning from Lessons**: Lin's work (1991, 1992, 1993) focuses on how $Q$-learning can scale up to more complicated tasks than those typically attempted, introducing several approaches for improving the learning rate of reinforcement learning. One

of the approaches combines apprentice learning and reinforcement learning into a method that relies on the trainer's ability to do the task in its entirety. In this approach, a human trainer performs the task occasionally, leading the agent through all the steps necessary to progress from a start state to a goal state. The sequence of (state, action, resulting state, reinforcement) quadruples that result from the human's actions is captured in a *lesson*. After a lesson has been recorded, the learning agent incorporates it into its experience by simulating the trainer's performance repeatedly. The agent does not need to perform the sequence of actions itself because all of the information necessary to do the $Q$-learning updates is present in the quadruples of the lesson.

The results of Lin's experiments indicate that the learners are able to learn more quickly with the trainer's aid than without it. In one case, the learner learns a task via the integrated approach that it does not learn with reinforcement learning alone. Unfortunately, like the trainer in apprentice learning, Lin's approach requires the human trainer to develop complete sequences of actions that are appropriate for teaching the learner. This approach does not address the issues of when the trainer's information is needed nor of how good the trainer needs to be in order to be effective.

**Learning via Interactive Teaching**: Clouse and Utgoff (1992) add a simple interface to a reinforcement learning approach to allow a human to interact with the automated learner on-line and in real-time. While the agent learns the multiple-step task via reinforcement learning, the human monitors the learner's performance and provides an action to the learner whenever she desires, providing the learner with a correct choice to make at that point in the task. At each time step, the learner senses whether the human trainer has supplied such an action. If so, the learner performs the trainer's action and gives itself an artificial positive reward for doing what the trainer suggested. Otherwise, the learner makes its own action choice based

on its developing policy. Because the learner receives an artificial reward, it learns to optimize a different set of rewards than if it did not interact with the trainer.

In an experiment with a complex control task, the learner performed the task after *two orders of magnitude less* training effort than that required with plain reinforcement learning, while receiving only an average of seven actions from the human trainer, each of which is a single key-stroke. This approach to integrating apprentice and reinforcement learning shows that the development of the policy can be *greatly* improved when a trainer attempts to teach the learner actively. However, the trainer decides when such training information is appropriate for the learner. Similarly to Lin's system, there is no sense of the best time for a trainer to provide information nor of the effects of the trainer's expertise on the results.

Figure 3.7 depicts the learner and trainer for these two systems, in which the trainer determines when to give the learner information. The learner can employ two meta-actions: perform its own action and perform the trainer's action. The learner's interaction policies, as in all systems discussed previously, are fixed. When the learner receives input from the trainer it must use it, and when the trainer does not provide the learner with help, the learner is on its own.

The trainers in these two systems decide whether to give the learner help or not. Unfortunately, both systems rely on a human trainer whose expertise at the task is unknown, and it is uncertain when the trainer should give information to the learner.

#### 3.3.2.2 Receiving Solicited Actions

Utgoff and Clouse (1991) show that the learner can take more control of the interactions between the trainer and learner, soliciting the trainer to supply actions. In the reinforcement learning component of their system, the learner updates its policy via a form of temporal difference learning: the value of the current state is changed to reflect the value of the successor state resulting from the application of the current action. The trainer's information is assimilated into the policy by inferring that the

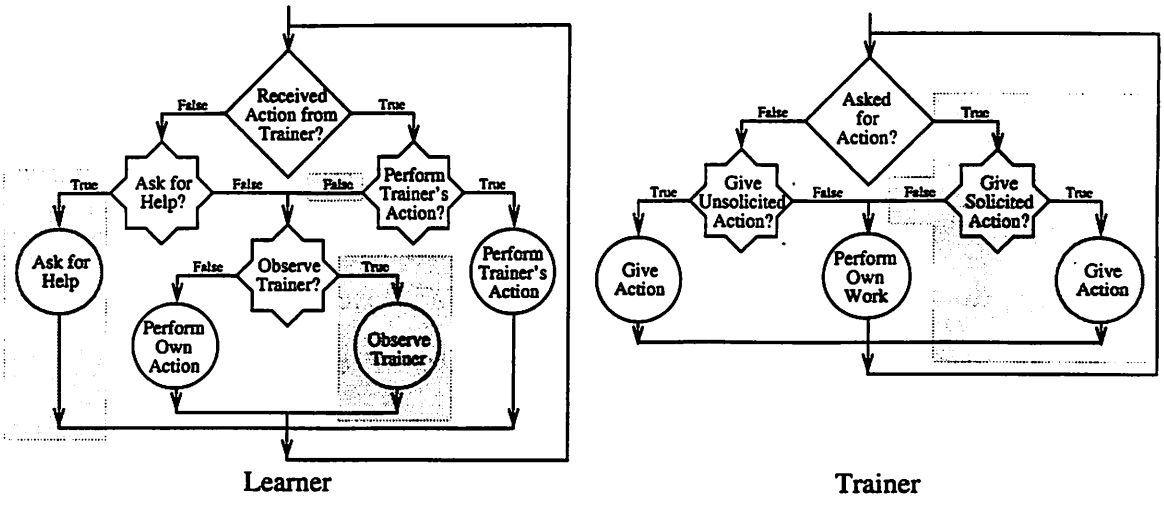



**Figure 3.7** Fitting Integrated Learning to the Learner/Trainer Model, where the trainer provides unsolicited actions

state resulting from the action chosen by the trainer should have a higher value than all of the other currently applicable states. As the learner trains with reinforcement learning, whenever the difference between the value of the current state and the value of the successor state exceeds a specific threshold, the automated trainer is queried and the trainer informs the learner of the correct action choice.

The learner is able to learn to perform a small version of the task after traversing it *only once* from start state to goal state, whereas an apprentice learning method requires two traversals and the reinforcement learning approach requires almost five hundred. For larger problems, the integrated method is still much faster than the other two individually. Furthermore, the frequency of requests for trainer's actions drops sharply as the learner improves its performance on the task.

Figure 3.8 depicts the learner and trainer for this system. Unlike all previous systems we have discussed, the interaction policy for deciding whether to ask the trainer for help is an automated policy with two possible outcomes. This interaction policy is based on the internal workings of the reinforcement learning algorithm and

indicates whether to ask the trainer for help or to perform a task action. All other interaction policies produce fixed values.

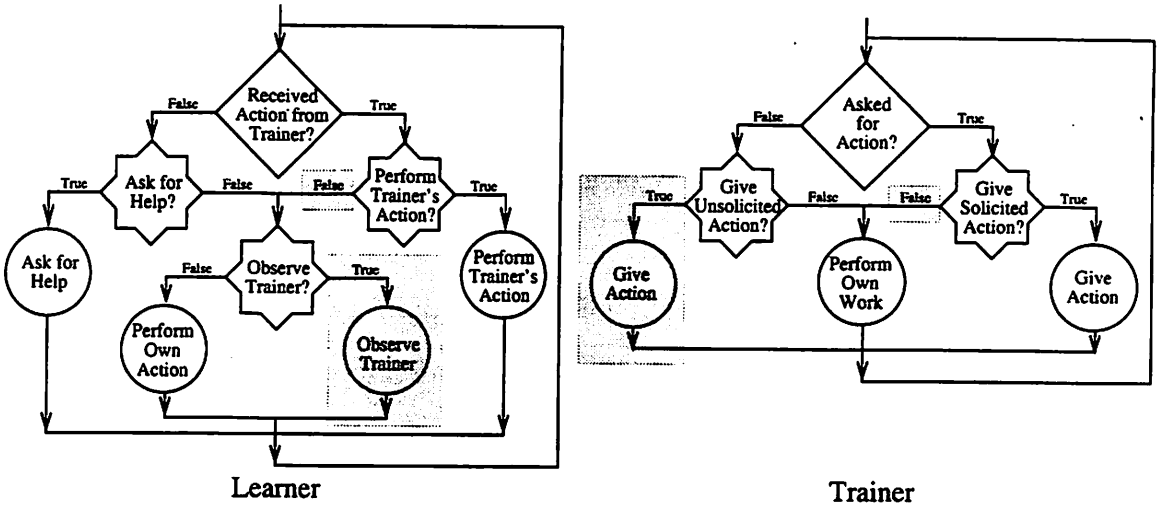



**Figure 3.8** Fitting Integrated Learning to the Learner/Trainer Model, where the trainer provides solicited actions

The learner in this method is more sophisticated than any previously mentioned because it can decide whether or not to ask the trainer for help. However, the trainer always provides the learner with perfect training information, telling the learner the optimal action to perform, and so this approach does not deal with the trainer's possible lack of expertise.

## 3.4 Other Related Work

Other systems have been built in which learners attempt to develop policies for multiple-step tasks. Like the integrated systems described above, these systems incorporate a human's knowledge in addition to evaluative feedback in learning to perform the task. The systems can be divided into two main groups. In one, the human provides high-level advice in the form of if-then rules. In the other, the human subdivides the task into smaller, more easily solved pieces.

### 3.4.1 If-Then Rules

In the integrated method described above, the trainer provides the learner with simple actions that should be applied to the current state. Another form of knowledge that the trainer can give the learner is general advice in the form of if-then rules. Unlike task actions, which only give information about the states with which they are paired, one piece of advice may be applicable to many situations that arise in the task.

In Maclin and Shavlik's (1996) Reinforcement and Advice-Taking Learning Environment (RATLE) system, while the agent learns its task via reinforcement learning, a human monitors the learner's performance. Whenever the human feels it is necessary, she provides the learner with advice in the form of an if-then rule that specifies a set of actions to take for all states that satisfy the rule's preconditions. Using techniques derived from knowledge-based neural networks (Towell, Shavlik & Noordewier, 1990), the learner incorporates the rule via changes to the multi-layer network that represents its policy. According to the structure of the rule, the learner adds nodes and connections to the network, and gives the connections initial weights. After the advice-induced changes to the network have been made, the learner continues learning, updating its policy based on reinforcement learning and possibly changing the values of the new weights.

When it is given advice, the learner achieves significantly higher levels of reward in performing the multiple-step task than when it receives no advice. The experiments also indicated that the time of arrival of the advice does not matter: whether given at the beginning or in the middle of learning, the learner is able to achieve the same high level of performance. One drawback of RATLE is that it requires the trainer to perform the following steps in giving advice: observe the learner's behavior, analyze that behavior, develop a rule to fix an observed difficulty, codify that rule in a manner understandable by the learner, and, finally, present the advice to the learner. This

is much more demanding on the trainer than having the trainer react naturally to the current situation by offering an action to perform, but, the advice may be more broadly applicable.

Gordon and Subramanian (1994) built a similar system in that the learner accepts rules from a trainer and then refines those rules autonomously. In their system, the learner relies on genetic algorithms (Holland, 1986), instead of reinforcement learning, to learn from task rewards. The trainer develops a set of if-then rules before the agent begins learning, codifying whatever knowledge seems appropriate. After the if-then rules are converted automatically into operational rules with the help of a database of domain knowledge, they become part of a population of rules that are continually refined by the genetic operators as the learner performs the task. The learner, when seeded with the trainer's advice, learns to perform the task better than learners without the advice. There is no real interaction between the learner and trainer, though, because the trainer can only give advice at the beginning of training.

### 3.4.2 Task Decomposition and Shaping

In the approaches discussed here, the underlying learning method is reinforcement learning. The apprentice learning aspect comes implicitly from the fact that a human either changes the definition of the problem as the learner progresses, training it on increasingly complex approximations of the required problem, or has spent a great deal of effort in defining the problem. The first approach is *shaping* and the next is *task decomposition*. In these two approaches, there is no direct interaction with a training agent. Thus, these approaches fit the learner/trainer model in the same manner as reinforcement learning, where the learner does not have the benefit of an on-line trainer. However, when one considers the extra effort of the human, these approaches fit the model similarly to many of the systems discussed above.

Shaping, which is a method from animal learning for training animals to perform complex motor tasks (Skinner, 1938), has been adopted recently for training auto-

mated learners (Gullapalli, 1992; Singh, 1992). The learner is trained on increasingly complex approximations of the required task, and would not be expected to learn to perform the task without this aid. In this case, the human expert specifies the approximations that allow the learner to develop its policy.

In task decomposition a human expert provides the agent with a set of subtasks to learn, thereby simplifying the overall learning problem (Mahadevan & Connell, 1992; Singh, 1992). The human expertise comes in the form of the *a priori* decomposition of the task into these subtasks, the priorities of the subtasks, and the identification of the subtasks to the learner. Without this task breakdown, the learner would not be able to develop an appropriate policy at all.

Task decomposition and shaping each take advantage of human expertise to lessen the difficulties of learning in multiple-step tasks. Unlike the previously discussed integrated methods, in these two, the trainer gives the learner domain knowledge that is much more complex than a simple task action.

## 3.5 Summary

The careful integration of apprentice learning and reinforcement learning will produce a hybrid that has many of the desirable characteristics of its constituents and thus will be better then either of them. Previous research has affirmed this statement partially, but has not addressed sufficiently other issues pertinent to integrating the two methods.

Combining the two methods raises a set of questions that must be addressed in designing, implementing, and applying a hybrid method. One must determine how the learner and trainer will interact, when the trainer will provide training actions to the learner, and how the learner will incorporate those actions into its developing policy. Furthermore, one must be concerned with how the trainer's expertise will affect the ability of the learner to develop a policy. In the next chapter, we present a

new hybrid system, ASK FOR HELP, that was designed to find answers to these and other questions.

# CHAPTER 4

# A NEW INTEGRATED SYSTEM

Each of the earlier systems that combine apprentice learning and reinforcement learning allows the learner to develop an effective policy more quickly than without the integration. Regardless of the accomplishments of these previous systems, there are still many fundamental questions to answer. Before integrated methods are widely adopted, we need to know, for example, when it is beneficial for the trainer to provide the learner with training actions. We must also ask how the trainer's expertise influences the learner's ability to acquire an appropriate policy, so that we can consider how to learn from trainers that provide occasional suboptimal actions.

In order to answers these and other questions, we have designed, developed, and implemented a new integrated approach, the ASK FOR HELP (AFH) approach. The other objective to satisfy in designing the AFH approach is to give it the desirable characteristics presented in Section 3.1.1 (page 34). These characteristics include the ability to learn from both the trainer and its own actions, and to learn an optimal policy. The extent to which these characteristics are actually present in the approach is a matter for empirical verification. ASK FOR HELP is also influenced by each of the earlier integrated systems.

This chapter continues by reiterating the research questions from Chapter 3 and discussing the requirements on a new integrated approach that allows us to find answers to those questions. We then discuss the new approach in detail.

## 4.1 Design Considerations

In Chapter 3 we pointed out several questions of interest to us that arise in integrating apprentice learning and reinforcement learning. To reiterate, the questions are:

1. How should the learner and trainer interact?
2. How should the learner incorporate the trainer's proffered information into its developing policy?
3. When is it effective for the learner to receive training actions?
4. To what extent does the expertise of the trainer affect the learning?

The design for our new integrated approach is motivated by our desire to answer these questions. Below, we show how the design of the new approach is influenced by each question. The specifics of the approach are given later, in Section 4.2.

The learner/trainer model described in Section 3.2 describes many of the interaction policies that a learner and trainer might employ in an integrated method. Because we are specifically concerned with determining when it is effective for the learner to receive training actions, we focus on the "Ask for Help?" interaction policy, which the learner uses to tell the trainer that it needs information. We have complete control over when the learner receives aid if the trainer's interaction policies require it to provide an action when asked and never allow it to give unsolicited responses. Thus, this is the setup of the new integrated approach. Part of our empirical study is aimed at determining whether our choices are effective.

We could also have developed a new approach in which the trainer determines when to give actions and the learner always performs the trainer's actions. This approach would also give us complete control over the receipt of trainer's actions, but we do not take it because one of the strategies for interaction that we wish to explore

is based on the learner's uncertainty about its current move choices. Thus, we place the learner in charge of acquiring the trainer's actions.

Care must be taken in designing the technique by which the learner assimilates the trainer's actions. The technique must allow the learner to update its policy based on the trainer's actions, but the technique must also allow the learner to develop an optimal policy. The technique we adopt is quite closely related to those in previous integrated systems.

There has been little work in determining when the learner should receive training actions. Most previous integrated systems allow the human trainer to provide the learner with actions whenever the human feels it is necessary, which does not provide much information about how to build and apply future integrated systems. Only one of the integrated systems discussed in the previous chapter addresses this issue directly (Utgoff & Clouse, 1991, Section 3.3.2.2 herein), having the learner decide when it should get actions from the trainer. Indeed, our new approach is quite similar to this previous one.

To aid us in answering this question, we implement two strategies for the "Ask for Help?" policy. One strategy is stochastic and the other depends on the learner's uncertainty in its current action choice. Each strategy allows us to look at how different amounts of trainer-supplied actions influence the learner's ability to develop an effective policy. The strategies also give us the opportunity to explore how the timing of the receipt of the actions affects the learning. Even though we are only dealing with the "Ask for Help?" interaction policy, the issue of when the learner should receive help is pertinent to any integrated system. Thus, whatever we discover will also be applicable to other approaches with differing interaction policies.

What we find out about the influence of the trainer's expertise will also apply to other integrated systems. Each of the previous integrated systems does not deal with this issue: they either assume that the trainer is perfect (Utgoff & Clouse, 1991)

or ignore the issue entirely (Clouse & Utgoff, 1992; Lin, 1992; Lin, 1993; Gordon & Subramanian, 1994; Maclin & Shavlik, 1996), relying on a human with unknown abilities. We wish to build a system in which we can vary the trainer's expertise directly and easily. The design decisions we make in setting up the interactions between the learner and trainer allow us to alter the trainer's expertise without changing the way in which the two agents interact. This question has great influence one the choice of problem domain for the empirical study. We need a domain in which we can build trainers whose expertise we can vary. As we describe in the next chapter, one of our domains meets this requirement nicely.

Considering each of the four questions presented at the beginning of this section has lead us to develop a new integrated approach. The main reason for introducing the new approach is to find answers to these questions. Of course, we also wish to determine how effective the approach is, and to determine the extent to which it outperforms apprentice learning and reinforcement learning.

## 4.2 The ASK FOR HELP Approach

Taking the issues of the previous section into consideration, we have developed the ASK FOR HELP approach. An overview of the ASK FOR HELP approach is presented in Figure 4.1, which depicts the components of AFH and their high-level interactions. Because AFH is an integrated approach, it consists of a task, a learner, and a trainer. The interactions between the learner and the task are identical to those in reinforcement learning: the learner observes task states, chooses actions to perform in the task, and receives rewards from the task. The interaction between the learner and the trainer is managed by the learner: the learner determines when the trainer should provide training actions. As depicted in the figure, the learner controls a switch that determines when it receives an action from the trainer. With the switch closed, the learner acquires an action, which it then performs. When the

switch is open, the learner is not requesting aid and will perform an action it itself has chosen. One can think of the trainer as producing an action for every state at every time step, where the actions are not used when the switch is open. This viewpoint is expensive in terms of the trainer's time. We prefer to think that by closing the switch the learner is asking for help (utilizing the "Ask for Help?" interaction policy) and that the trainer only provides an action when asked.

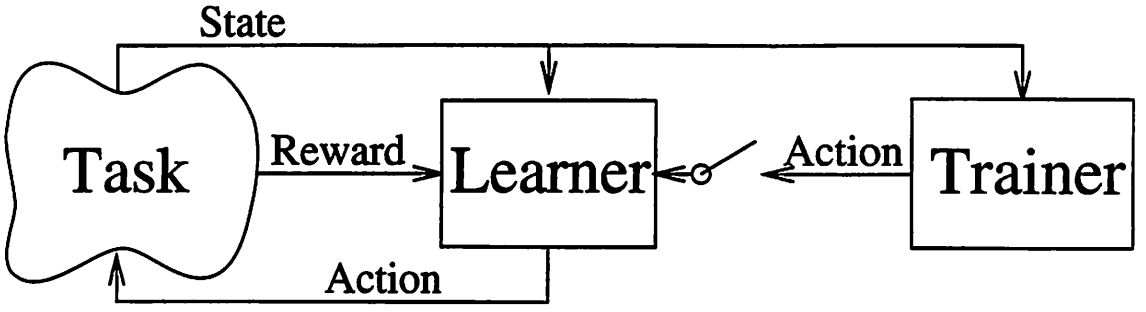



**Figure 4.1** Overview of the ASK FOR HELP Approach

### 4.2.1 The Learner

The learner in ASK FOR HELP, like the more general learner described in Section 3.2.1, attempts to develop a policy that informs the agent of which action to perform in any given state. Also like the more general learning agent, this agent has access to two sources of knowledge: rewards and the actions offered by the training agent. The learner is specified completely by its learning algorithm, the manner in which it incorporates the trainer's actions into its developing policy, and its interaction policies.

#### 4.2.1.1 The Learning Algorithm

The choice of policy update algorithm is based on each of the desirable characteristics presented in Section 3.1.1. The learner should be able to adapt based on scalar

rewards, to explore its environment, and to develop a policy that optimizes a measure of the rewards received. As discussed in Section 2.3.3, $Q$-learning was designed with each of these characteristics in mind. Thus, the AFH learner's main learning mechanism is $Q$-learning. The specifics of the parameter settings and the exploration policy will be discussed with other experiment details, in the next chapter. Although other reinforcement learning algorithms could have been chosen, we use $Q$-learning because of its widespread use and ease of implementation. Because the trainer's input is a form of knowledge not exploited in any of the current reinforcement learning algorithms, all such algorithms will benefit from the input of a training agent.

#### 4.2.1.2 Incorporating the Trainer's Actions

In addition to the characteristics mentioned above, the learning agent must also be able to adapt based on actions chosen by the trainer. The problem that must be addressed is how to incorporate an action into $Q$-learning, which was designed to learn from scalar rewards. Furthermore, in learning from the trainer's actions, the learner must not be precluded from learning a policy that optimizes a measure of the rewards received. Stated differently, the learner must not attempt to mimic the trainer at the expense of learning an optimal policy.

In AFH we implement a straightforward technique to solve this problem. The learner assimilates each trainer's action by first performing the action, just as though the learner itself had chosen it. Then, after receiving whatever reward is appropriate for performing that action, the learner updates its policy via $Q$-learning; again, just as though the learner itself had produced the action. Thus, the learner incorporates the action and its consequences into its experience without the need for any changes to $Q$-learning. The trainer's action is treated similarly to an action that the learner might choose for exploration purposes. It is not based on the learner's policy, but the learner performs it anyway and learns from its consequences.

This incorporation technique is similar to those in previous integrated systems, lending credence to its efficacy. It is like the technique that Lin employs (Section 3.3.2.1), although a major difference between that approach and AFH is that Lin requires the trainer to perform the entire task, whereas AFH adapts according to individual actions from the trainer. Clouse and Utgoff (1992) also have the learner perform the trainer's actions. They provide an additional artificial positive reward upon which the learner bases its $Q$-learning update, "rewarding" the learner for performing the trainer's action. Unfortunately, this approach raises the possible problem of learning to act like the trainer at the expense of optimizing the rewards received.

#### 4.2.1.3 Interaction Policies

The learner's interaction policies determine how it acquires information from the trainer and what it does with that information. Like previous integrated systems, these interaction policies are more limited than in the general learner/trainer model. In particular:

- The learner will always perform the trainer's action when there is one, and
- The learner will never choose to observe the trainer.

Thus, two of the three learner's interaction policies are constant, as shown in Figure 4.2. "Perform Trainer's Action?" is true, and "Observe Trainer?" is false. Because our research examines when the learner should receive training actions, the interaction policy "Ask for Help?" does not produce a constant value. The learner determines the output of this interaction policy, deciding whether to ask the trainer for an action or to choose its own action.

In our empirical study, we compare two strategies for determining when to ask the trainer for aid. The first strategy, which we call the *uniform asking strategy*, is stochastic: the learner asks for help randomly throughout training. The second

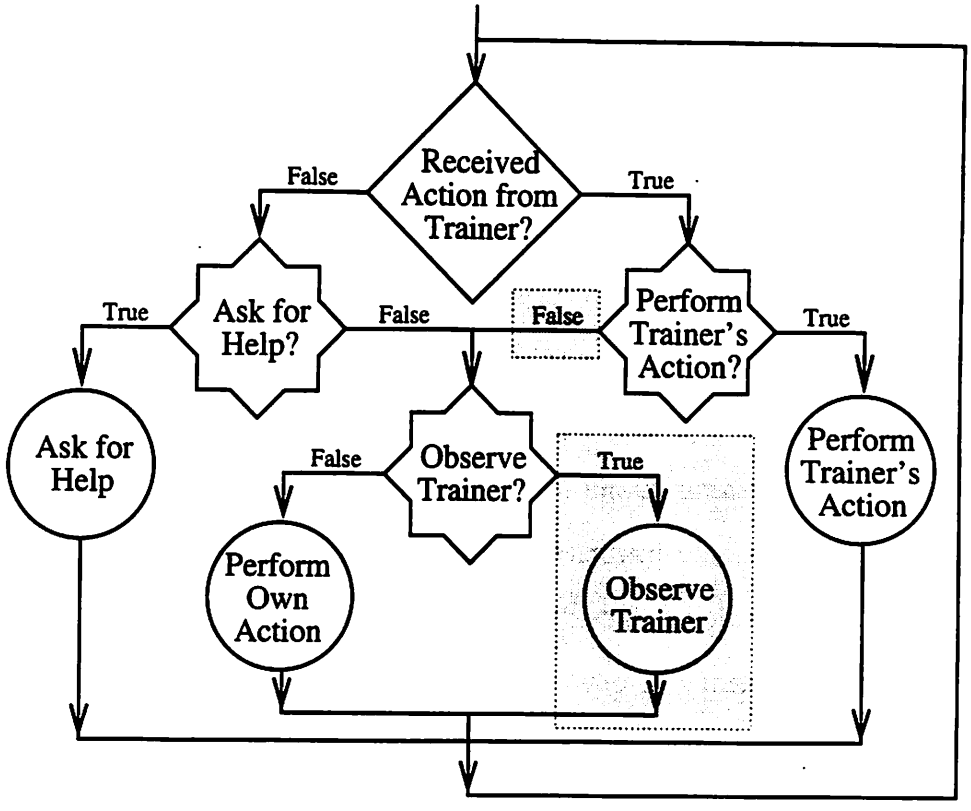



**Figure 4.2** The ASK FOR HELP Learner

strategy, the *uncertainty asking strategy*, takes a more sophisticated approach, basing the decision on the learner's uncertainty about its current move choices.

**Uniform Asking Strategy**: The objective in developing the uniform asking strategy is three-fold. First, we wish to determine whether the AFH approach shows improvements over its constituent methods. We also desire to observe the extent to which trainer's actions influence the learning rate. The final objective is to establish a performance level for a learning agent in an integrated system. This performance level can then be compared against the performance achieved with other asking strategies. If a learner with a new asking strategy performs more poorly than with this simple one, then, clearly, the new strategy is not worthwhile.

In the uniform asking strategy, the "Ask for Help?" interaction policy is stochastic. The learner's queries are spread uniformly throughout training, based on a parameter that establishes the percentage of time steps on which the learner asks for help. This *asking rate* parameter specifies the probability that the learner will ask for an action on any given time step. There is an entire family of "Ask for Help?" interaction policies specified by this strategy, one for each possible asking rate.

**Uncertainty Asking Strategy**: To study further when the learner should ask for help, we also develop a more sophisticated asking strategy. This strategy is based on the learner's uncertainty about its current action choices. The goal is to develop a strategy that reflects our intuitions about when human learners require instruction on a problem and to determine the extent to which the timing of the receipt of trainer's actions affects the learning rate.

Certainly, humans benefit from help when in novel situations. Indeed, humans seek help when they are confused or otherwise unable to decide upon a course of action, even if they have experienced the situation in the past. It is difficult to specify exactly when humans are unsure because we cannot examine directly a person's thinking processes. This is not the case for an automated learner, though. We can base our determination of the learner's uncertainty on its current decision policy, relying on the $Q$-values of its current action choices. We have developed a test that decides whether the learner is unsure of its choices, and whether the learner should ask for help. The test indicates the need for help when all of the action choices have similar $Q$-values. If the minimum and maximum values are close enough to each other (which is specified by a parameter we call the *asking factor*), then the intervening values must also be similar. Thus, when the learner must choose one of its options, it can not make a clear distinction between them.

Examining the interval between the extreme values identifies only situations in which the learner is clearly uncertain. Another form of uncertainty may arise when

only the top two or more choices have similar values. Our approach will not have the learner ask for help in this situation because it may be the case that the two choices are truly similar. Furthermore, by only asking when the extremes differ by a preset amount, the learner will ask the trainer for aid less frequently than if it bases that decision on only the highest values. The uncertainty asking strategy only requires the learner to ask for help when it is clearly uncertain of its choices, as determined by the asking factor.

The asking factor controls how conservative the learner is. With a small factor, the learner is infrequently uncertain, but with a larger factor, the learner asks for aid quite often. In the experiments described in the next chapter, we alter the asking factor in order to analyze the possibly different learning behaviors.

### 4.2.2 The Trainer

The trainer in AFH is automated. The main reason for doing so is to have complete control over the trainer's behavior—something that we cannot guarantee with human trainers. We also rely on automated trainers instead of humans so that we can perform many experiments easily.

The training agent attempts to help the learner develop its policy, and is depicted in Figure 4.3. Whenever the learner requests an action, the trainer evaluates the current state and makes a decision about the action it would perform, which it then passes on to the learner. The two interaction policies are constant, causing the trainer to give the learner an action if one is requested and to ignore the learner otherwise.

One of the issues we wish to study is how the expertise of the trainer influences the ability of the learner to develop a policy. In the experiments described in the next chapter, we take direct control of the trainer's expertise, building a set of trainers whose proficiency varies. Some of the trainers can produce the optimal action to take and others choose actions randomly a pre-specified percentage of the time. With these trainers we can determine the importance of the trainer's problem-solving abilities.

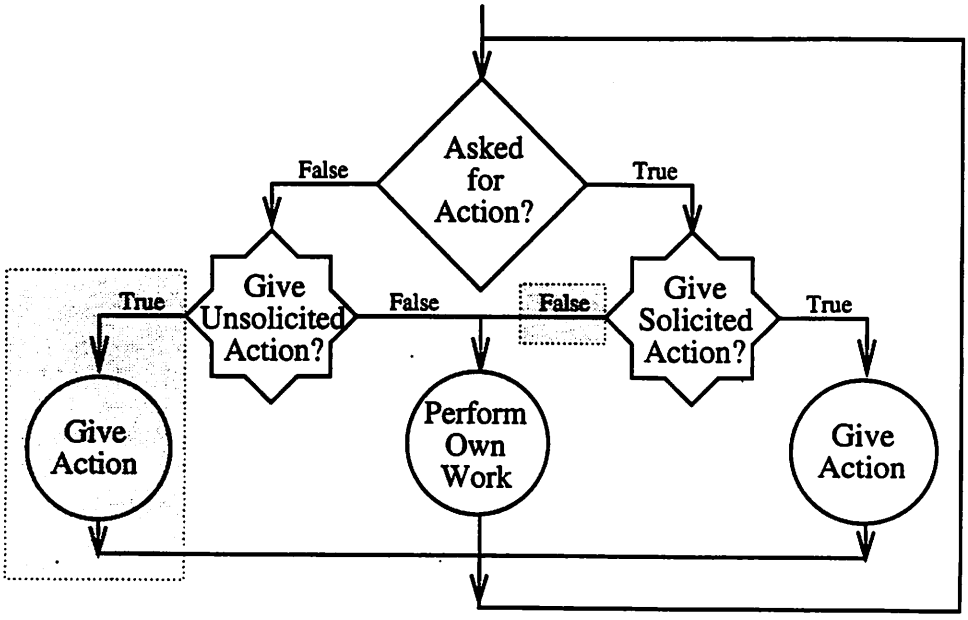



**Figure 4.3** The ASK FOR HELP Trainer

# CHAPTER 5

# EMPIRICAL STUDY

In Chapter 4 we hypothesize that ASK FOR HELP is an effective approach to integrating apprentice learning and reinforcement learning. More importantly, AFH allows us to ask, and then answer, fundamental questions about integrating the two learning methods. The questions we wish to answer and upon which we base our design of AFH are:

1. How should the learner and trainer interact?
2. How should the learner incorporate the trainer's proffered information into its developing policy?
3. When is it effective for the learner to receive training actions?
4. To what extent does the expertise of the trainer affect the learning?

In order to answer these general questions about integrated methods, we run four experiments, each of which is designed to address a particular question about the ASK FOR HELP approach. Because we choose to implement AFH with particular interaction policies and incorporation technique, the experiments will provide insight into the answers for the first two questions above. The experiments address the next two questions more directly.

In the next section, we describe the experiment domains in detail. Then, beginning in Section 5.2, we present the four experiments, describing the experimental design and the results for each, and discussing their implications. Finally, we end the chapter with a summary of the results and answers to the general questions listed above.

## 5.1 Experiment Domains

In the experiments described below, we employ two domain tasks, each with different characteristics. The first domain is maze navigation, which is a Markovian decision task. The maze problems in our study, which can also be considered grid-world problems with stationary obstacles, can be solved via dynamic programming. Thus, we can find optimal policies upon which to base training agents. We have direct control over the size and complexity of the mazes, so we can alter these factors in order to examine how they affect the performance of AFH. Maze tasks are a member of a class of discrete-state problems, such as game-tree search and other tasks that can be cast as graph traversal problems. Although the mazes are not representative of all discrete-state problems, due to their limited branching factor and moderate-sized state-spaces, they are sufficiently complex to be difficult problems to solve. Therefore, the conclusions we draw based on the empirical study will be applicable to many other domains.

The second domain, a race track domain in which a simulated car drives on a track, serves as a test for the generality of the conclusions we draw about the maze results. With this domain, we determine how well our approach scales to a larger problem with different characteristics. The race track domain differs from the maze domain along at least four dimensions. First, it is in continuous space. Second, because we do not discretize the space, it requires a generalization structure, such as a neural network (Rumelhart & McClelland, 1986), to store its policy. Third, the state of the task is represented in terms of sensors mounted on the car, so many task states will map to the same sensor representation. Thus, it is not a Markovian decision task. Finally, one cannot find an optimal policy via dynamic programming because of the size of the state-space, and so the trainer must be built by other means and may not be optimal.

### 5.1.1 Maze Tasks

The maze domain is graph-traversal in the form of two-dimensional mazes, which are a type of Markovian decision task. Maze problems are representative of a class of multiple-step tasks that have discrete state-spaces and few transitions out of each state. Other problems with these characteristics include some game-tree search and constraint satisfaction problems. Mazes are a simplification of larger, more complex multiple-step tasks, which may have continuous state-spaces and many transitions out of each state. Even so, maze tasks are difficult to solve, and results obtained with these tasks will generalize to many other multiple-step tasks.

For the particular mazes employed in our study, in each maze cell, the learner can choose one of four actions: up, down, left, or right. When performing an action that is blocked by a wall, the agent does not move. The objective of the learner is to traverse the maze optimally from the start cell to the goal cell. The learner receives its only non-negative reward, 1.0, when it enters the goal cell.

The experiments are performed with both deterministic and stochastic mazes. For the deterministic mazes, each action is performed as specified: for example, an up action will lead to the cell that is above the current cell. For stochastic mazes, each action has a 25% chance of being changed to an action at a right-angle to itself, half the time resulting in a right turn and the other half, in a left turn. For example, performing an up action will most often move the learner into the cell above its current location, but it will sometimes move the agent into the cell to the left or right. This is representative of other stochastic problems where an action can lead to one of a set of states, probabilistically.

In our study, we employ a series of mazes whose sizes range from a 20x20 maze with only 230 cells, up to a 200x200 maze with 29,329 cells. Even though the size of these state-spaces is small compared to other problems (such as Chess), our experiments indicate that they are still challenging problems to solve based on the amount of

training required by the learner to develop an appropriate policy. The 20x20 maze is depicted in Figure 5.1. The maze walls are shown as cells with a hashed pattern. The start cell is in the top-left corner and is marked with an 'X', and the goal cell is in the bottom-right corner, marked with a star.

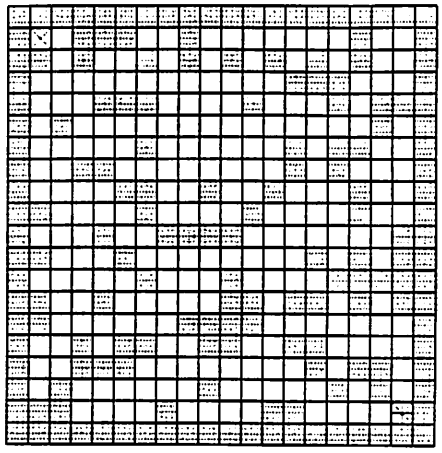


**Figure 5.1** The 20x20 maze, with 230 cells. The start cell is at the top left of the maze, marked with an "x," and the goal cell is at the bottom left corner.

There are at least four advantages to employing maze tasks for the empirical study. First, we can generate maze problems of varying sizes, enabling us to explore how our results scale to larger problems. Second, we can find the optimal policy for our moderately-sized mazes via dynamic programming. Both the construction of the trainer and the definition of the stopping criterion for the experiments exploit this fact. We can also store the developing policies in simple tables, rather than relying on not well-understood generalization mechanisms, because the particular mazes we use are discrete and moderately small. Finally, discreteness and moderate size allow us to record certain information that we cannot for other tasks, such as which states are visited and how frequently. Through analysis of this type of data, we will be able to gain knowledge about how a trainer's influence affects the learner's performance.

Although the issue of generalization is interesting, we avoid it in the study of the maze tasks because it introduces several confounding factors, clouding some of the issues we wish to explore. In recognition of the importance of the issue, we do deal with it in the experiment described in Section 5.4, in which the problem task relies on neural networks (Rumelhart & McClelland, 1986) to store the policy. This gives us an opportunity to determine how well AFH scales and how much it depends on the tabular representation.

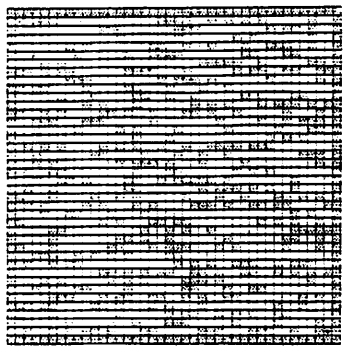
#### 5.1.1.1 Maze Creation

Producing the actual mazes for the experiments is a straightforward task of deciding on the dimensions of the maze, filling in the border walls of the maze, and then filling in 25% of the cells randomly with walls. After this process completes, the mazes must be inspected by hand to ensure that all cells are reachable from the start cell in the top-left corner. When cells are isolated, we either remove or relocate walls. The resulting number of states is just the number of cells that do not have walls in them. Depictions of the physical layouts of the mazes are in Figure 5.2.

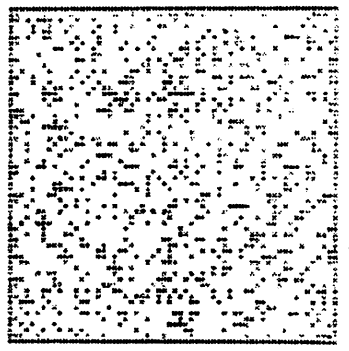
#### 5.1.1.2 Learning Algorithm

In all of the experiments in the maze tasks, the learner develops its policy via $Q$-learning, described in Section 2.3.3. The $Q$-functions for each of the four actions are stored in separate tables. The $Q$-value for a particular state and action, $Q(s, a)$, is just the value in the $s$th location of the table for action $a$.

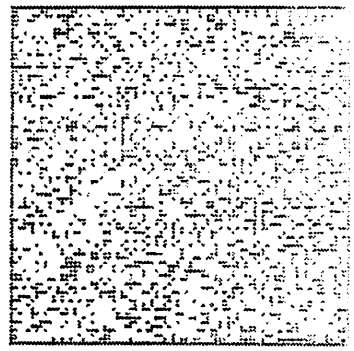
The policy defined by the $Q$-functions is based on the $Q$-values and on a random factor to facilitate exploration. A large portion of the time (95%), the learner performs the action whose $Q$-function has the highest value for the current state. During the other 5% of the time, the learner chooses an action uniformly from among the four. Although the exploration setting may affect how quickly the learner converges on an optimal policy, not enough is known about exploration to suggest a setting that is



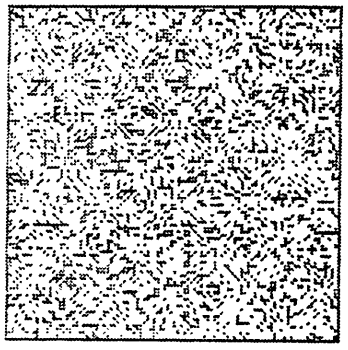
The 40x40 Maze
1,018 cells



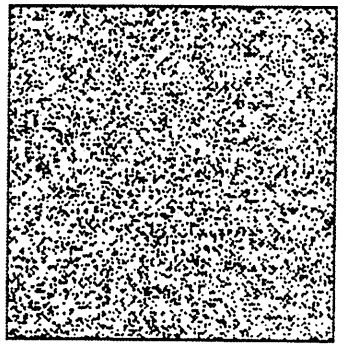
The 60x60 Maze
2,530 cells



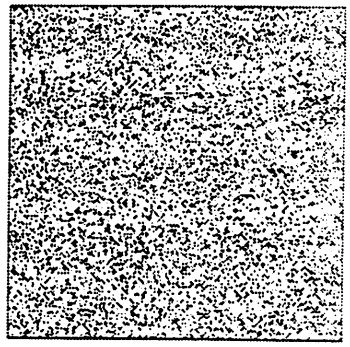
The 80x80 Maze
4,268 cells



The 100x100 Maze
6,661 cells



The 150x150 Maze
16,473 cells



The 200x200 Maze
29,329 cells

**Figure 5.2** The six other mazes

obviously best. We choose the 5% rate in order to make the learner choose at least one random action when following the optimal path in the smallest maze. Of course, for the larger mazes, this means the learner might choose many random actions in traversing the maze. To offset this, the stopping criterion for the experiments takes into account the exploration factor, as described in Section 5.2.1 below.

For every experiment with the maze tasks, the algorithm's parameters are: $\alpha = 0.15$ (step size), and $\gamma = 0.99$ (discount factor) (See Equation 2.7, page 26). No attempt was made to optimize these parameters for any of the experiments. We

believe this setting for the step size parameter helps the learner propagate information quickly without causing the $Q$-values to grow too quickly. The setting of the discount factor causes the learner to consider how its current choices will affect it in the distant future. We choose this setting mainly because the learner only receives a reward at completion of the task.

#### 5.1.1.3 Trainers

As we pointed out earlier, an advantage of discrete mazes is that we can determine the optimal policy for navigating through each maze–at least for moderate-sized mazes. To build a trainer, we need only determine the optimal policy, which we do easily via dynamic programming. In the experiments, the trainers respond to the learner's queries by giving an action that is based on the optimal policy for that particular maze. Some trainers always suggest an optimal action, but others give random actions occasionally.

### 5.1.2 Race Track Task

This domain is based on the Race Track game (Gardner, 1973), and, in its discrete-state form, has been a test-bed for other research in reinforcement learning (Barto, Bradtke & Singh, 1995). The task simulates a car navigating on a track, which can be of any length and shape, and has a start line at one end and a finish line at the other. This task is representative of simple robotics navigation tasks, where the robot must move from its current location to a goal location and where the robot relies on on-board sensors to determine its location in the world. An example track, which is one of the tracks employed in the experiments, is displayed in Figure 5.3.

The problem is for the learner to drive the car from the start line to the finish line quickly and without running off of the track. At each time step, the agent first observes its state, relying on a set of sensors for information about its current situation. Then, the agent decides whether to accelerate, decelerate, or turn left

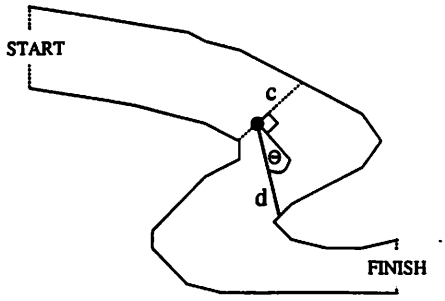



**Figure 5.3** Example race track

or right. After performing an action and changing the state of the task, the agent receives one of three reinforcement signals. When the car drives off of the track the reward is $-1.0$, and when the car crosses the finish line it receives a reward of 1.0. If neither of these conditions holds, the learner is assessed a small negative reward ($-0.01$) at each time step. Thus, the objective of the learner is to avoid running off the track and to reach the finish line with as few actions as possible. The learner knows nothing about the dynamics of the car on the race track.

#### 5.1.2.1 Sensors

The car is equipped with five sensors that measure its position and velocity on the track. Each sensor reports a continuous value, except for $f$, which is boolean. The sensors give the agent the following information:

1. Speed of the car, $s$,
2. Distance to the edge of the track that is directly ahead of the car, $d$,
3. Relative position of the car between the left and right edges of the track, $c$,
4. Angle of the car with respect to the center-line of the track, $\theta$, and
5. Whether the finish line is directly ahead, $f$.

The first sensor measures the speed of the car, which is determined simply by the distance the car traveled in the last time step. Another sensor measures the distance to the edge of the track that is directly ahead of the car. This measurement is depicted in Figure 5.3 as the line labeled $d$. If the car were to continue on its present heading, it would follow that line. This sensor has an outside range of fifty units; anything over fifty is reported as fifty.

A third sensor measures the relative position of the car between the left and right edges of the track. When the car is in the center of the track, it reports a zero value. As the car approaches the left edge of the track, the value increases, reaching one when the car drives off of the track. The sensor reports negative values when the car is closer to the right edge than the left edge, reaching negative one when the car drives off the right edge. The dashed line labeled $c$ in the figure represents this measurement. That line segment is the shortest line segment containing the car that has endpoints on the left and right edges of the track. The sensor reports the relative position of the car on that line segment.

The fourth sensor measures the angle of the car with respect to the left and right edges of the track. If the car is driving directly down the center of the track, this sensor reports a zero value. As the car turns to the left, the sensor reports increasing angles, reaching 90 degrees when the car is headed directly at the left edge and continuing to 180 degrees as the car continues to turn until it is heading back up the track. Turning to the right causes the sensor to register negative angles, from zero degrees for straight ahead to $-90$ degrees when driving directly at the right edge, to $-179$ degrees when driving in the wrong direction. That angle is labeled $\theta$ in the figure and is measured with respect to the perpendicular of shortest line between the right and left edges going through the car.

The last sensor informs the car whether the finish line is directly ahead or not. If the finish line is ahead of the car, but outside of the fifty unit range, the sensor cannot "see" the finish line, and reports a value of zero.

#### 5.1.2.2 Representation

The state of the car on the track is captured by twenty features. The first five features are just the measurements of the five sensors: the speed of the car $s$, the distance straight ahead to a wall $d$, the relative position of the car with respect to the edges of the track $c$, the angle of the car $\theta$, and whether the car can "see" the finish line $f$. The next fifteen features are the quadratic terms of these five, i.e. $s^2, sd, sc, s\theta, sf, d^2, dc, d\theta, df, c^2, c\theta, cf, \theta^2, f\theta$, and $f^2$.

The features are scaled so that their values are in $[-1.0, 1.0]$ in order to aid with the rate of training (Hampson & Volper, 1986). The range of the car's speed is $0 \leq s \leq 10$, so scaling is done by subtracting five and dividing by two. The angle's range is $-\pi < \theta \leq \pi$; dividing by $\pi$ gives the scaled range. As the sensors have a limited range of fifty units, $-1 \leq d/25 - 1 \leq 1$. The other two features, $c$ and $f$ are already in the required range, so no scaling is necessary. The quadratic terms will be in the required range because they are products of the scaled first-order terms.

#### 5.1.2.3 Learning Algorithm

The learner employs $Q$-learning to develop its policy, and stores its $Q$-function in a set of weight vectors, one for each of the four actions. The $Q$-value for a given state and action, $Q(s, a)$, is given by the inner product of two vectors, the weight vector for the action $a$ and the feature vector representing $s$, which is composed of scaled sensor values and their quadratic terms, as described above. In developing its policy, the learner adjusts the weights in the weight vectors via a variant of Equation 2.7 (page 26) that includes the TD($\lambda$) approach (see Equation 2.4, page 23).

In the race track task, we employ a more sophisticated exploration strategy than in the maze task. The policy defined by the $Q$-function is based on the Boltzmann distribution of the $Q$-values: the next action, $a$, is randomly chosen based on the probabilities given by

$$\forall a \in \text{actions},\ P(a) = \frac{e^{Q(s,a)/T}}{\sum_{b \in \text{actions}} e^{Q(s,b)/T}}$$

where $s$ is the current state, $T$ is the temperature parameter that is reduced over time to diminish the randomness of the policy, and "actions" is the set of actions the learner can perform in the current state. When the trainer does not give an action to the learner, the learner chooses its action based on the probabilities defined by the equation above. When the trainer provides an action, the learner does not check its policy, but performs the trainer's action.

The parameter settings in the experiments are: $\alpha = 0.35$ (step size), $\gamma = 0.75$ (discount factor), $\lambda = 0.75$ (TD($\lambda$) parameter). The temperature parameter decays slowly from $T = 5.0$ down to its lowest value, $T = 0.2$, via a linear factor of 0.9995. We did not attempt to optimize any of the parameter settings for these experiments. Unfortunately, there is no principled approach to use when picking these parameter values. We choose the step size parameter based on previous experience with linear networks that suggests that the 0.35 rate facilitates quick learning, but not at the risk of having the weights grow without bound. The $\lambda$ setting is also chosen based on previous experience. Because mapping many states to the same representation (as discussed above) introduces uncertainty into the problem, we choose the setting for the discount factor in order to have the learner focus on how its actions affect its near-term future. The three temperature settings are chosen to facilitate exploration throughout the training and are also based on practical concerns about numeric overflows when calculating the action probabilities.

#### 5.1.2.4 Trainer

The trainer that instructs the learning agent is automated and capable of performing the task at a moderate level of expertise. The trainer does not produce optimal actions because not enough is known about this task to produce such a trainer easily. Although the trainer does not always perform an optimal action, it can drive the car from each track's start position to the finish line.

The trainer was built by hand, via a cyclic design-code-test approach. First, a moderate amount of knowledge engineering, of examining different race track situations and ascertaining the action that a good driver should take, determined the initial trainer. Then the trainer was tested on the track. When the trainer crossed a track boundary, the code was redesigned and re-tested. This process continued until the trainer could drive on both the tracks used in the experiments, not only from every start location, but also from other, randomly selected points on the track.

The objective of the trainer, as it drives, is to remain on the track and progress towards the finish line quickly. Whenever the trainer is in a situation in which it can accelerate, it will do so. Otherwise, the trainer either turns to avoid crossing a track boundary and to stay close to the center of the track, or slows down. The trainer does not produce an action when the car is driving in the wrong direction on the track. The trainer's algorithm for choosing an action is given in Table 5.1.

#### 5.1.2.5 Simulation

The simulation for the domain is based on the one described by Gardner (1973). The track is stored as two series of line segments, one for the left edge and one for the right edge as well as a line segment for each of the start and finish lines. The state of the car is represented internally to the simulation as its previous and current positions on the Cartesian plane.

**Table 5.1** Rules for the hand-coded trainer. The quantities in the if-tests are based on the car's current state, and are given after the algorithm.

```
IF the finish line is directly ahead OR the wall ahead is really far away
      THEN accelerate
ELSIF the car is heading down the center of the track
      THEN
            IF the wall ahead is far enough away
                  THEN accelerate
            ELSIF the wall ahead is moderately close
                  THEN decelerate
            ELSIF the car is closer to the left wall than the right wall
                  THEN turn right
                  ELSE turn left
            ENDIF
ELSIF the car is heading too much to the right
      THEN
            IF the wall ahead is moderately close
                  THEN decelerate
                  ELSE turn left
            ENDIF
ELSIF the car is heading too much to the left
      THEN
            IF the wall ahead is moderately close
                  THEN decelerate
                  ELSE turn right
            ENDIF
      ELSE do nothing, because the car is heading backwards
ENDIF
```

- The finish line is directly ahead: $f = 1$
- The wall ahead is really far away: $d > ((s+4)(s+1))$
- Heading down the center of track: $-40° < \theta < 40°$
- The wall ahead is far enough away: $d > ((s+4)(s+1))/2$
- The wall ahead is moderately close: $(4s(s+1)/10) < d < (11s(s+1)/16)$
- Heading too much to the right: $-90° < \theta < -40°$
- Heading too much to the left: $40° < \theta < 90°$

When the car accelerates or decelerates, the speed is incremented or decremented by one, with a lower speed limit of one and an upper speed limit of ten. The car's current speed is just the distance it traveled in the last time interval. The new position of the car is:

$$x(t+1) = x(t) + (s + \Delta s)(x(t) - x(t-1))/s$$

$$y(t+1) = y(t) + (s + \Delta s)(y(t) - y(t-1))/s$$

$$\Delta s = \begin{cases} +1 & \text{if accelerating and } s < 10 \\ -1 & \text{if decelerating and } s > 1 \\ 0 & \text{otherwise} \end{cases}$$

where $(x(t+1), y(t+1))$, $(x(t), y(t))$, and $(x(t-1), y(t-1))$ are the new, current and previous locations of the car, and $s$ is the previous speed of the car (the distance between $(x(t), y(t))$ and $(x(t-1), y(t-1))$).

The change in heading is based on the way in which one of Gardner's cars would turn. If the car's current speed is $s$, the change in heading is based on a right triangle whose legs are $s$ units and one unit long. The change in heading is the size of the angle opposite the side of one unit length.

$$x(t+1) = x(t) + \frac{\Delta\alpha(y(t) - y(t-1)) + s(x(t) - x(t-1))}{\sqrt{s^2+1}}$$

$$y(t+1) = y(t) + \frac{s(y(t) - y(t-1)) - \Delta\alpha(x(t) - x(t-1))}{\sqrt{s^2+1}}$$

$$\Delta\alpha = \begin{cases} +1 & \text{for a left turn} \\ -1 & \text{for a right turn} \end{cases}$$

If the line segment joining the previous location of the car and the current location of the car crosses any of the line segments making up the edges or crosses the start line, the car has driven off of the track. If that same line segment crosses the line segment designating the finish, the car has crossed the finish line.

## 5.2 Training with an Optimal Trainer

This first experiment is designed to determine the extent to which asking for help randomly from an optimal trainer speeds the learning process. A secondary

objective is to determine whether AFH with the uniform asking strategy is effective at integrating apprentice learning and reinforcement learning.

We find that the learner does improve its learning rate via AFH. Generally, the more often the optimal trainer provides actions, the more quickly the learner acquires an optimal policy. Although this result is not surprising, we also find that more interaction with the trainer does not necessarily improve the learner's learning rate. The experiment also reveals that little trainer interaction is necessary in order to improve significantly over not having a trainer at all.

### 5.2.1 Experimental Design

For this first experiment, the learners attempt to navigate in each of the different mazes (both deterministic and stochastic), and the trainers always suggest optimal actions. Whenever several actions are optimal, the trainer chooses one randomly. There are seven deterministic mazes and four stochastic mazes[1]. For each of the eleven mazes, there are eight individual sub-experiments in which we vary the learner's asking rate, which determines how often the learner asks for an action from the trainer. For example, when the parameter is set at 10%, the learner asks the trainer for an action 10% of the time and chooses its own action 90% of the time. The asking rate ranges from 0%, which is equivalent to standard $Q$-learning, up to 100%, where the learner always asks for the trainer's actions (a form of apprentice learning).

Each individual sub-experiment consists of ten runs, each of which begins with the entries in the $Q$-function tables set to zeroes. A run ends when the learner meets the stopping criterion of performing at least 95% optimal actions when traversing the maze ten times from the start to the goal. Remember that the learner cannot perform 100% optimal actions because it chooses a random action 5% of the time. A run is also stopped, but considered a failure, when either the total number of

[1]Because preliminary experimentation revealed that the learners required prohibitive amounts of time to learn to navigate in the larger stochastic mazes, those mazes are not used in this study

actions exceeds fifty million, the total number of actions in a single trial exceeds one million, or the total number of trials exceeds ten thousand. These three cases are designed to limit the amount of time it takes to run the experiments, and are based on results from preliminary experimentation. These bounds affect the runs for the larger deterministic mazes and for all of the stochastic mazes.

Each run consists of many trials, all of which begin with the learner in the start cell of the maze (top-left corner) and end when the learner reaches the goal cell (bottom-right corner). The stopping criterion for the run is tested at the end of each trial. If the criterion is met, the run stops; otherwise, the run continues with the beginning of a new trial, placing the learner back at the start cell.

For each run we record the total number of actions the learning agent performs in the run (which also includes the actions suggested by the trainer), the total number of times the trainer is asked for aid, and the total number of trials in the run. Because the learner asks for aid uniformly, the number of times the trainer presents an action is just the product of the asking rate and the total number of actions performed. We also record the frequency with which each maze cell is visited is certain runs.

In this experiment we want to examine the amount of improvement the learner achieves in learning rate when using AFH. In order to say that the learner acquires an optimal policy more quickly via AFH than via $Q$-learning, we need to find statistically significant reductions in the mean number of actions required to learn an optimal policy when asking the trainer for aid compared to using $Q$-learning alone. The results for every level of asking rate are compared to the 0% rate, which is standard $Q$-learning. We perform one-tailed student's t-tests and seek significances at the 0.01 level. In the sequel, when we report a statistically significant difference in means, the level of significance will always be at least 0.01. Any significant reductions we find will indicate that the AFH approach is better than its $Q$-learning component. We also want to determine whether AFH performs better than apprentice learning, so we

compare results at all asking rates to the 100% rate (which is a form of apprentice learning).

### 5.2.2 Results

Figure 5.4 shows a graph that depicts the results for the 40x40 maze. The results for the remaining mazes are presented later. The data from which these graphs are derived are presented in the Appendix. The learner's asking rate is presented along the independent axis, and the average number of actions necessary to achieve the stopping criterion is presented on the dependent axis. To facilitate comparison with a learning agent that does not have access to the trainer (at the 0% asking rate), the dashed line labeled "Q-learning" is extended from the dependent axis, with two parallel lines that represent the 95% confidence interval for that value. The curve labeled "Learner" depicts the number of actions required by the learner to achieve the stopping criterion for different levels of asking rate. Each point on that curve represents the mean number of actions for a particular setting of the asking rate parameter, and is shown with its 95% confidence bar. Most of the confidence bars are too short to see. Although the figure shows the results as a continuous curve, note that the data is only gathered at the discrete points shown.

The results for the 40x40 maze indicate that the trainer's instruction reduces the amount of training necessary to achieve the stopping criterion. With standard $Q$-learning the learner requires an average of slightly over 1.8 million actions to learn an adequate policy. With only a 1% asking rate, the mean number of learner's actions drops to just over 1.2 million, which is not statistically significantly different from the number of actions required with $Q$-learning alone, due to the high variance. However, even when asking for aid only 5% of the time, the number of actions necessary to achieve the stopping criterion is significantly lower. The mean numbers of actions for all asking rates higher than 5% are significantly better than those for not asking. At each asking rate starting at 50% and continuing to 100%, approximately 30,000

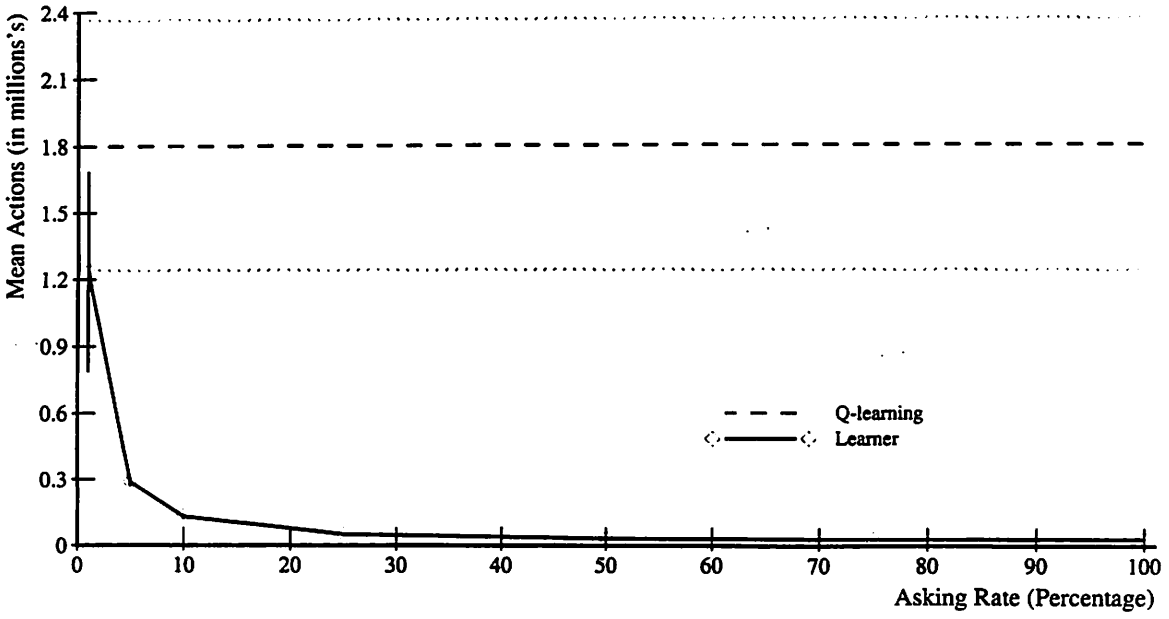



**Figure 5.4** Results for the 40x40 deterministic maze with the optimal trainer. The 95% confidence bars are shown for each point, although some are too small to see. Starting at the 5% asking rate, the means are statistically better than those achieved with $Q$-learning alone. The means at the 50% and 75% rates are statistically indistinguishable from those at the 100% rate.

actions are required to solve the problem—a reduction by a factor of sixty from $Q$-learning alone. In fact, there is no statistically significant difference between the result at the 50% asking rate and the 100% rate, and there is no significant difference between the 75% rate and the 100% rate.

As shown in Figure 5.5, the results for the other deterministic mazes are similar. The asking rate is depicted along the x-axis, the maze size along the y-axis, and the z-axis represents the mean number of actions (in million's). The surface shows the mean number of actions required to learn an appropriate policy for each of the asking rates and mazes. The previous graph (Figure 5.4) is a slice through this graph at the maze size of 40.

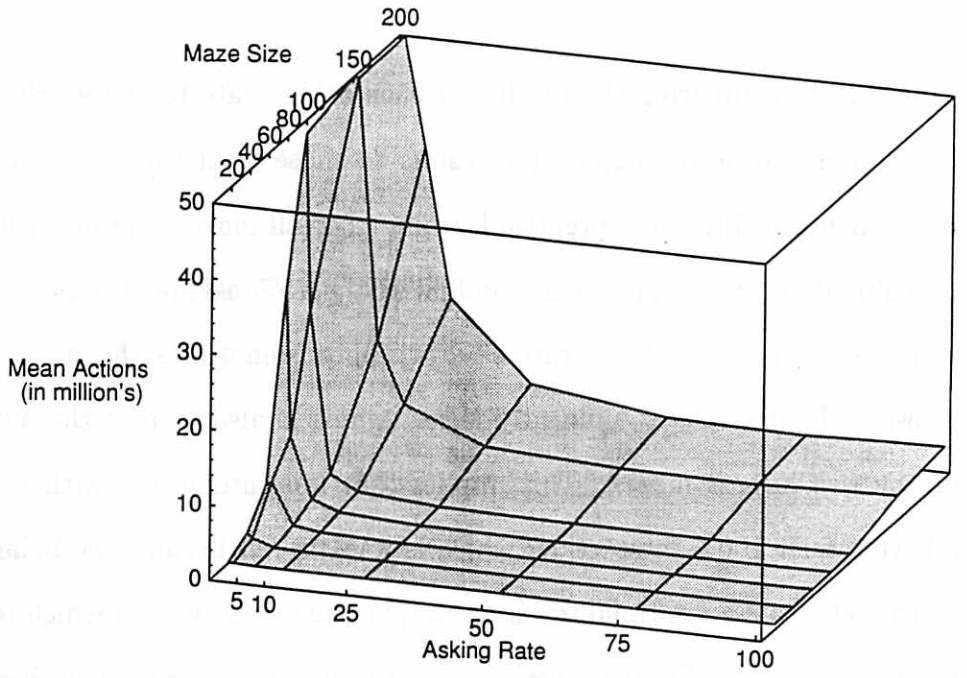



**Figure 5.5** Results for all deterministic mazes with the optimal trainer. All means at the 5% rate and higher are statistically better then those achieved with $Q$-learning alone (the 0% asking rate), for all mazes. The means at the 75% asking rate are either no different from or better than the means at the 100% rate, also for all mazes.

For every maze, the learner requires the most actions with $Q$-learning alone, depicted on the left-hand face of the graph at the asking rate of 0%. For the two largest mazes—200x200 and 150x150, with 29,329 and 16,473 states, respectively—the runs with $Q$-learning alone were terminated as failures. This is also true at the 1% level of asking for both of these mazes. For the 100x100 maze (6,661 cells), only one of the ten runs with standard $Q$-learning succeeded; all other $Q$-learning runs were terminated for this maze before an appropriate policy was learned.

For all of the mazes except the two largest and for all asking rates higher than 1%, the mean number of actions required to meet the stopping criterion is statistically significantly lower than that needed with $Q$-learning alone. Stated differently, AFH

performs significantly better than $Q$-learning alone for all asking rates higher than 1%.

In addition to comparing the results for each asking rate to those achieved with $Q$-learning alone, we also compare the results to those achieved when asking all of the time, which is a form of apprentice learning. For all mazes, the mean number of actions required to obtain an optimal policy at the 75% asking rate is never worse than that required at the 100% rate. Surprisingly, sometimes the mean achieved at the lower asking rate is significantly *lower*, which is also true at the 50% asking rate for the two largest mazes. This implies that less interaction with the trainer can be better than 100% interaction, which is unexpected because we believed that learning entirely from an optimal trainer would be the fastest way in which to acquire an optimal policy. This finding gives support to our earlier statement that gaining action-selection experience over a range of states is preferred over training exclusively on the solution path.

The results for the four stochastic mazes are quite similar to those for the deterministic mazes. The main difference is that the runs for all settings of maze and asking rate take longer to complete in the stochastic mazes than in the deterministic mazes. In fact, many runs with the stochastic mazes were terminated because they were requiring too much time. For all the stochastic mazes, all standard $Q$-learning runs were terminated as failures before the learners achieved optimal policies. Thus, we cannot make a statistical comparison between $Q$-learning and AFH for these mazes. The runs were also terminated for the asking rates of 1% and 5% for all mazes, and the 1%, 5% and 10% asking rates for the two larger mazes, the 60x60 and 80x80 mazes. However, the learners that asked for aid at least 25% of the time are quite effective at developing an appropriate policy. Thus, the learner is able to acquire an optimal policy with AFH when it does not do so via reinforcement learning alone (given the experiment constraints). For all four mazes, the learner performs significantly bet-

ter at the 100% asking rate than at any other asking rate. Thus, for these mazes apprentice learning is more effective than an integrated approach (with an optimal trainer).

Figure 5.6 depicts the results for the stochastic mazes. When comparing this figure to Figure 5.5 note that the scale on the z-axis is considerably different. The general observation from the deterministic maze results still holds: as the asking rate increases the mean number of actions decreases (moving left-to-right on the graph).

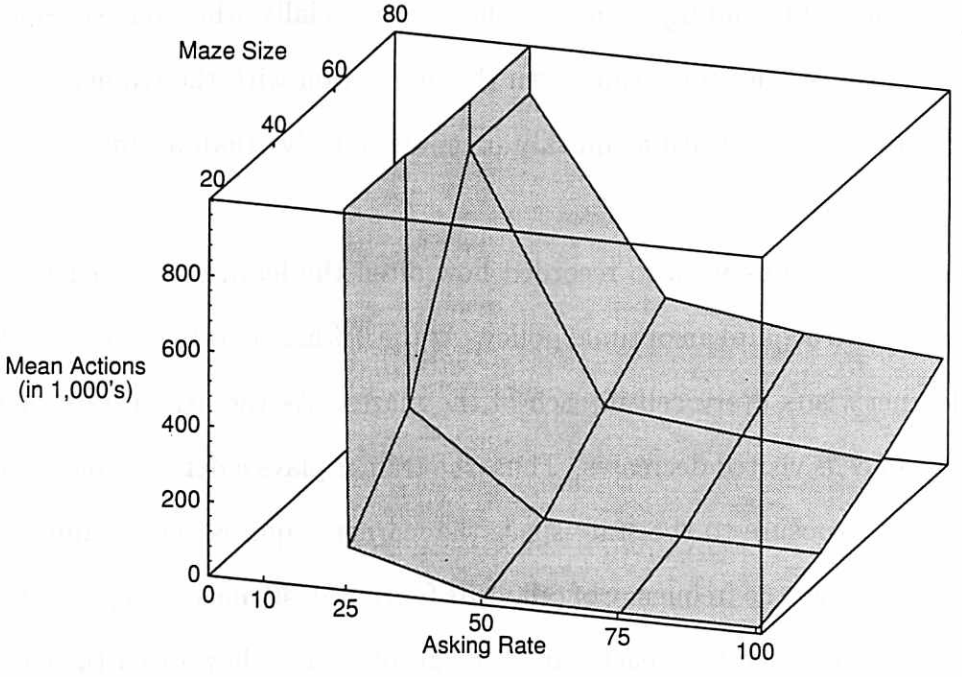



**Figure 5.6** Results for all stochastic mazes with the optimal trainer. All runs at the 0%, 5% and 10% asking rates were terminated because they had exceeded the preset maximums. Even so, these results resemble those from the deterministic mazes closely.

### 5.2.3 Discussion

Based on these results, we conclude that the ASK FOR HELP approach is effective for integrating apprentice learning and reinforcement learning, at least when

the trainer provides optimal actions and that AFH can perform better than its constituents. We also conclude that more help is generally better than less help. However, it is not necessary to receive aid all of the time from the trainer: We find that the results for the 75% asking rate are never significantly worse than those at the 100% asking rate for all the deterministic mazes. In fact, the results at the lower asking rate are sometimes better. That is, receiving aid three quarters of the time is the same (or better than) receiving it all of the time. Moreover, for the two largest mazes, the learner learns more quickly when receiving aid half of the time versus receiving it all of the time. This finding is quite promising, especially when one considers the cost of having the trainer. One can lessen the interaction with the trainer, at a reduction in expense, but still learn as quickly (or more quickly) than asking the trainer more often.

For certain runs we also recorded how often the learner visits each maze cell in attempting to acquire an optimal policy. At the 0% asking rate (standard $Q$-learning), the learner visits every cell in each of the mazes. As the asking rate increases, the number of cells visited decreases. Thus, the trainer plays a definite role in determining how much exposure to the state-space the learner acquires. For example, the left of Figure 5.7 shows the frequency of cell visits for the 40x40 maze on a particular run with $Q$-learning. The height of each bar in the graph depicts how often the corresponding maze cell was visited, relative to the maximum for the maze. The start cell is at the left corner of the graph, and the goal cell is in the right corner. For this run, every maze cell was visited at least once by the learner. The right of Figure 5.7 depicts the frequency of cell visits for the 40x40 maze on a particular run at the 50% asking rate. Most of the learner's experience was acquired in a particular area of the graph (depicted with the highest bars), but the learner had also spent a small portion of its time exploring other parts of the maze. For this particular run, the learner visited

only 450 of the 1,018 maze cells, a reduction of over half. This attests that the learner can visit fewer states than $Q$-learning requires, yet still learn an optimal policy.

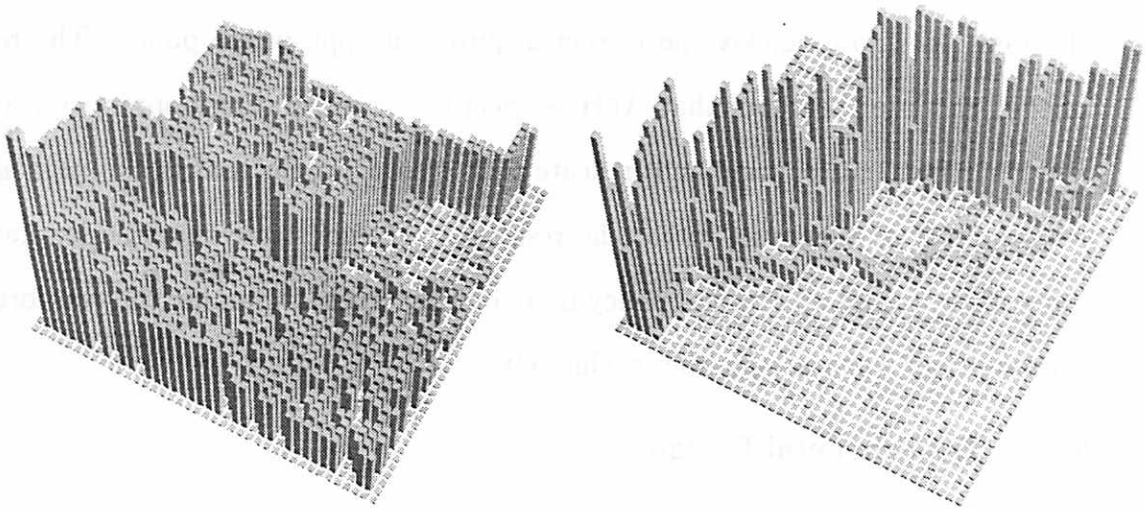


**Figure 5.7** Frequency of cell visitation for the 40x40 deterministic maze on particular runs. The maze on the left represents the cells visited when $Q$-learning alone was used. The right maze represents a run in which the learner asked the trainer for aid half of the time. All cells of the maze were visited when $Q$-learning was used, but only 450 of the 1,018 cells were visited when the trainer was used. The highest bar in the left maze indicates 7,528 cell visitations, whereas the highest in the left maze is only at 628.

An informal analysis of the frequency data for various runs and mazes suggests that the amount of exposure the learner receives is roughly determined by the asking rate. With a low asking rate, the learner visits almost all of the maze cells. As the asking rate increases, the learner visits fewer and fewer cells. Thus, there seems to be a relationship between the increase in learning rate and the decrease in state-space exposure. Further experimentation needs to be performed in order to determine whether the reduction in the number of cells visited is truly related to the reduction in the total number of actions required to learn an optimal policy.

## 5.3 Training with Suboptimal Trainers

This experiment is designed to determine the extent to which the expertise of the trainer affects the learning process. We note that the better a trainer is at performing the task, the more quickly the learner acquires an appropriate policy. The results further validate the claim that AFH is effective for integrating apprentice learning and reinforcement learning and indicate that even a trainer that is suboptimal will help improve the learning rate. The results also demonstrate that the learner can sometimes acquire an optimal policy more quickly when combining the two forms of learning than when using either exclusively.

### 5.3.1 Experimental Design

As in the first experiment, for this one we employ each of the different mazes (both deterministic and stochastic) and vary the asking rate from 0% up to 100%. Unlike the previous experiment, we also vary the trainer's expertise: the trainer does not always suggest an optimal action. The trainer's ability to choose optimal actions is controlled by an error parameter that determines how often the trainer returns a suboptimal action. For example, when the error parameter is set at 25%, the trainer returns a suboptimal action 25% of the time. With six settings of the error parameter, from 10% up to 75%, we are able to see how the trainer's expertise influences the rate at which the learner acquires its policy. We do not test the 100% error rate because preliminary experiments indicated that the runs would have taken prohibitive amounts of time, even for the smallest maze. A 100% error rate means that the actions given by the trainer are always the wrong actions; that is, the trainer's actions lead the learner away from the goal continually.

Each individual sub-experiment, with a particular maze, asking rate, and error rate, is performed in the same manner as in the previous experiment. There are ten runs, each of which begins with zeroed $Q$-values and ends when either the learner

meets the stopping criterion or surpasses the preset action or trial limits. Each run consists of several trials in which the learner attempts to navigate from the start cell to the goal. Because we vary the maze, the learner's asking rate, and the trainer's error rate, we ran 294 separate sub-experiments.

In this experiment we want to examine the amount of improvement the learner achieves in learning rate when using AFH and relying on different trainers. We perform statistical significance tests on the difference between the results achieved for $Q$-learning alone versus those at each of the asking rates and error rates. Because we also want to determine whether AFH performs better than apprentice learning, we compare results at all asking rates to the 100% rate (which is a form of apprentice learning). Like the previous experiment, we perform one-tailed t-tests and report significance at the 0.01 level.

### 5.3.2 Results

Figure 5.8 depicts the results of this experiment for the 40x40 maze, leaving off the curve for the 75% error rate, which is shown below in Figure 5.9. The numerical data for this experiment are presented in the Appendix. The graph gives the learner's asking rate on the independent axis, and the mean number of actions to end a run successfully on the dependent axis. Each curve on the graph represents the different settings of the error parameter. The curves for $Q$-learning and for learning from an optimal trainer are present for comparative purposes.

The results depicted in the figure indicate that even suboptimal trainers are capable of speeding the learning process significantly. Starting with the 10% asking rate, all results for all error rates are statistically better than standard $Q$-learning. The results for the 5% asking rate are also significant for the 0%, 10%, and 25% error rates. Note also that each of the curves is bounded above by a curve with a higher error rate and below by a curve with a lower error rate, for almost all of the asking rates. Thus, there is a gradual degradation of performance as the trainer's error rate

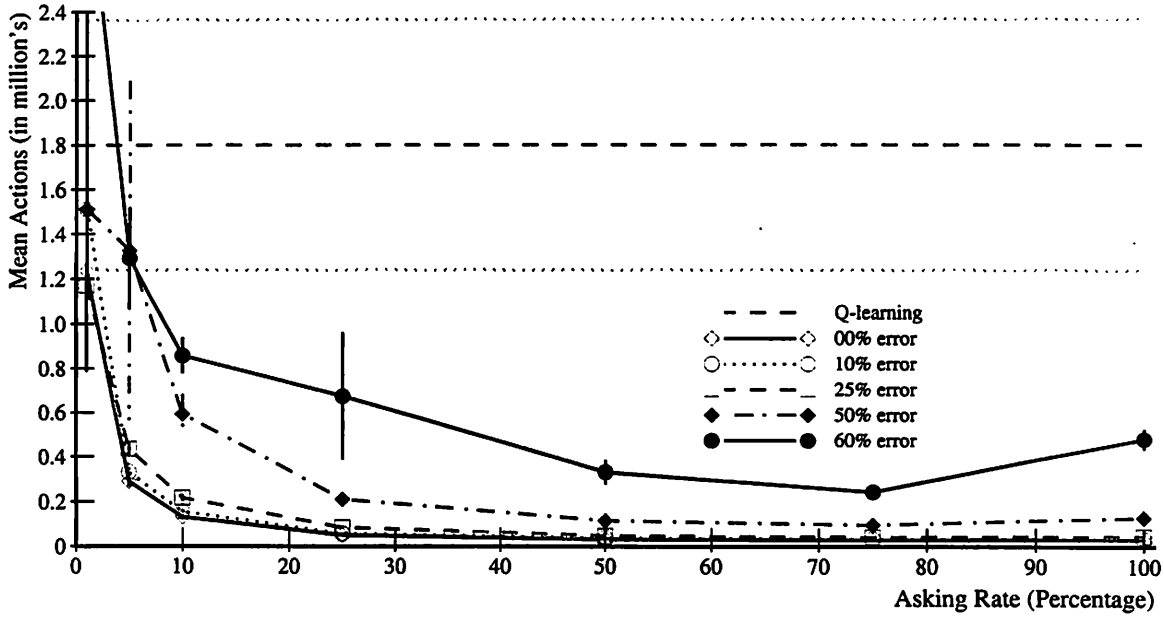



**Figure 5.8** Results for the 40x40 deterministic maze with all trainers. The 95% confidence bars are shown for each point. Starting at the 10% asking rate, all means are statistically better than those achieved with $Q$-learning alone. The means at the 75% rate are never worse than and are sometimes better than those at the 100% rate.

increases. Also notice that there is little difference between the results with the optimal trainer versus with the trainers that are incorrect 10% and 25% of the time. Thus, AFH is better than $Q$-learning alone for various settings of the asking rate.

AFH is also better than apprentice learning alone for certain asking rates. For all levels of trainer error, the results obtained at the 75% asking rate are never worse than those at the 100% rate. In fact, for the error rates of 25%, 50%, and 60%, the learner learns significantly faster at the 75% rate than when it relies totally on the trainer. For the 60% error rate, there is an astonishing difference between the mean number of actions required at the 75% asking rate and those required at the 100% asking rate: the learner acquires an appropriate policy in close to *half* the number of actions. The means observed at the 50% asking rate are also significantly lower

at the 50% and 60% error rates. These findings further the claim that an integrated approach can be better than apprentice learning.

In addition to the results for standard $Q$-learning and the optimal trainer, Figure 5.9 shows the results for the trainer that chooses a suboptimal action 75% of the time. At the 1% asking rate there is no significant difference between $Q$-learning and this learner, but even at the 5% asking rate the learner takes significantly longer to find a good policy. At the 10% asking rate the learner takes almost four times longer to find a good policy. All other runs at the 75% error rate were terminated as failures because they were taking too long. Although previous results show that a suboptimal trainer can still help the learner improve its learning rate, these results demonstrate that the trainer must have a minimal level of expertise on the problem.

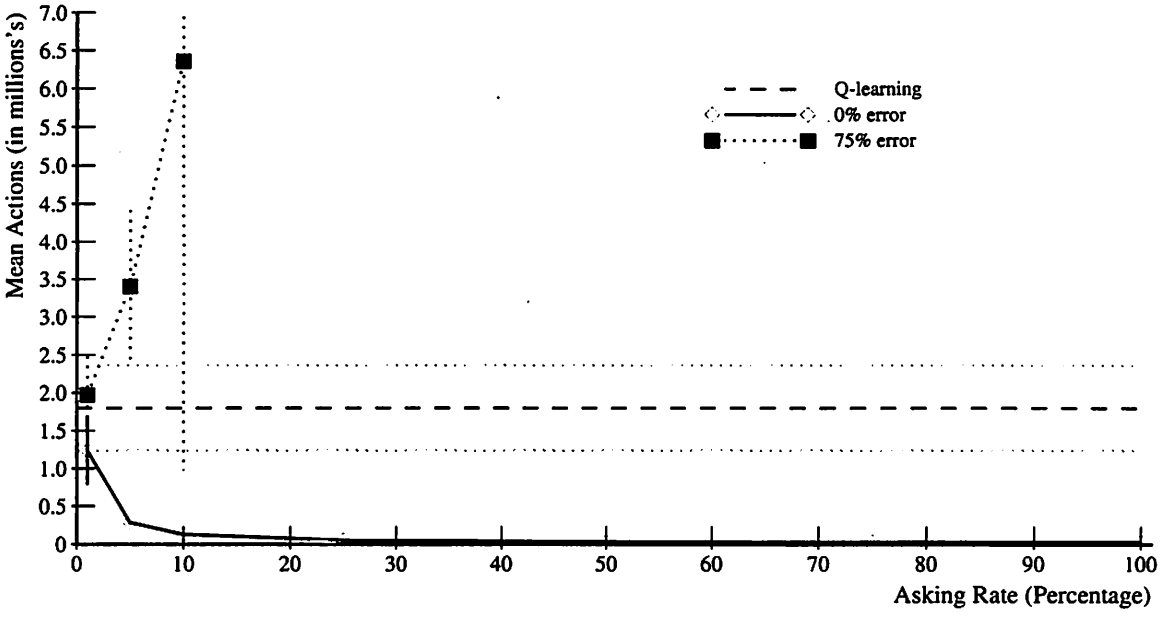



**Figure 5.9** Results for the 40x40 deterministic maze with a trainer that is incorrect 75% of the time. The means are never better than, and sometimes significantly worse than, not having a trainer at all. Runs with asking rates higher than 10% were terminated because the preset limits had been exceeded.

The results for all the other mazes also support the finding that a suboptimal trainer can be effective. The results for the 80x80 maze are shown in Figure 5.10, and those for the 200x200 maze are in Figure 5.11. For the 80x80 maze, the results are quite similar to the 40x40 maze, except that more mean actions are required at every asking rate, which should be expected because the 80x80 maze is larger than the 40x40 maze. For the 200x200 maze, many of the runs did not complete. For example, none of the sub-experiments with the 60% error rate completed except for the one at the 75% asking rate. All other runs in all other sub-experiments were terminated because they had surpassed the preset limits.

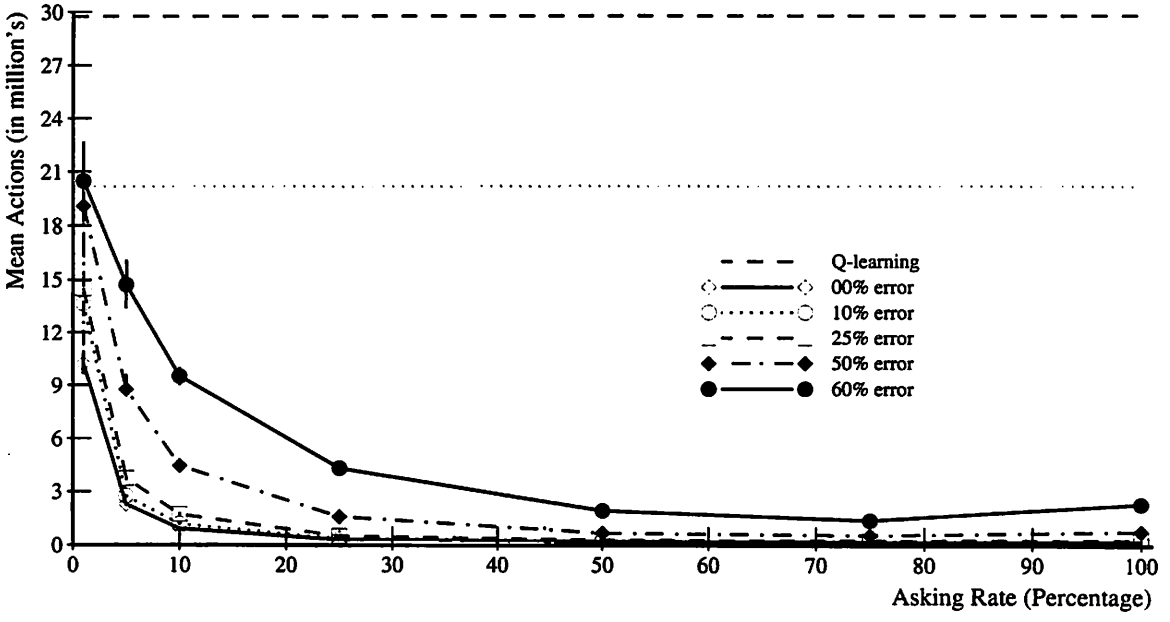



**Figure 5.10** Results for the 80x80 deterministic maze with all trainers. Note the similarity between this graph and Figure 5.8.

These two graphs also show that the 75% asking rate produces the best performance, at least for the higher error rates. As we noted before, the 75% asking rate is the only setting under which the learner finds an optimal policy for the largest maze

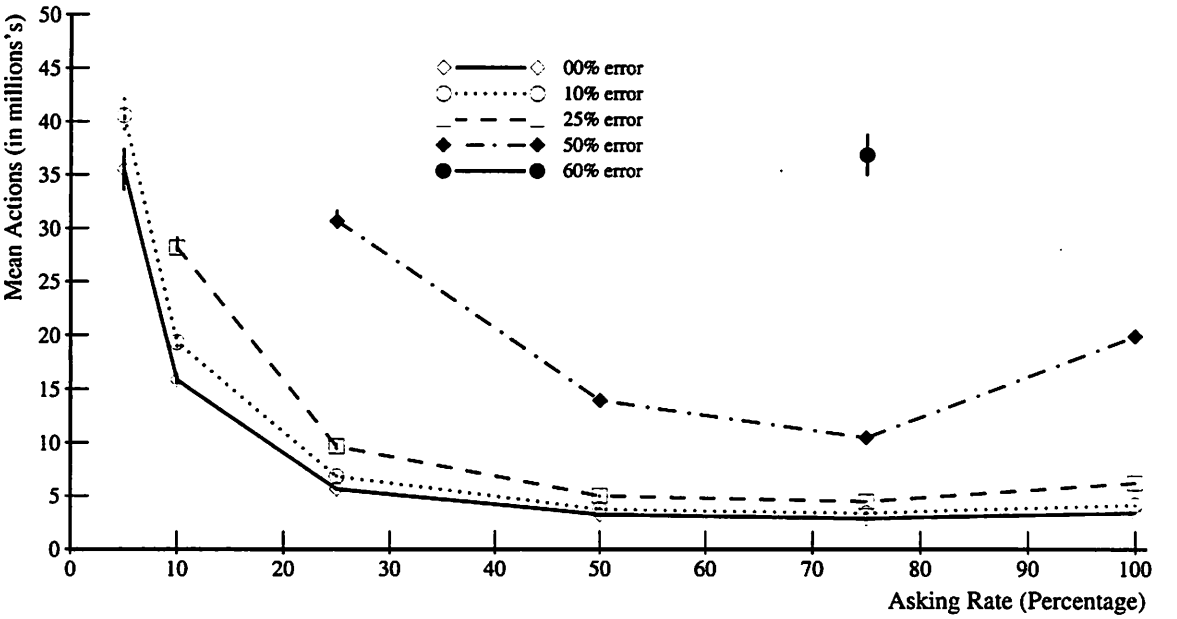



**Figure 5.11** Results for the 200x200 deterministic maze with all trainers. Note that the only runs that completed successfully with the 60% error trainer are at the 75% asking rate. Runs at other error rates were also terminated at the lower asking rates.

and a trainer that is incorrect 60% of the time. It is obvious from the graph that the 75% rate also leads to finding an optimal policy quickest for the 50% error rate. Closer analysis of the data for the 200x200 maze also shows this to be true for every error rate, including the 0% rate. As the maze size decreases, this effect is diminished, but even for the smallest maze, the 75% rate can produce better results than asking all of the time. This suggests that when applying AFH to other problems, one may wish to choose the 75% asking rate. This observation is born out in the next experiment.

### 5.3.3 Discussion

These results further validate the claim that AFH is an effective integration of apprentice learning and reinforcement learning. The learner learns more quickly with

the trainer's aid, even when the trainer is suboptimal. In answer to the question of how much the trainer's expertise influences learning, we conclude that the better a trainer is, the more the learner can benefit from the trainer's actions. However, the trainer does not need to be perfect in order for the learner to improve with the trainer's help. Furthermore, a trainer that misleads a learner frequently will cause the learner to take longer to acquire an effective policy, possibly even precluding the learner from doing so. Thus, the trainer's expertise does influence the learner's performance directly, and, in applying AFH to other domains, one must be aware of the trainer's expertise.

The results of this experiment indicate that relying on a trainer that is frequently incorrect is detrimental to the learner. In situations in which the learner acquires information from a trainer whose expertise is unknown, one would like the learner to be able to detect that its trainer is not proficient enough to be of help. A possible technique is for the learner to observe its own performance level and then ignore the trainer when it notices that its performance is no longer improving. Because this problem may occur in situations where a human serves as trainer, it is worth examining further.

One can also view the results of this experiment in terms of the number of optimal actions given by the trainer versus the number of suboptimal actions given. Although the experiment was not designed to give us this information, it can be determined from the data we did gather. Specifically, the asking rate parameter and error rate parameter control the percentage of optimal actions that the trainer gives to the learner. For example, at the 25% asking rate and the 10% error rate, 22.5% ($= 25\% \times (100\% - 10\%)$) of all of the actions performed by the learner are known to be optimal, 2.5% are known to be suboptimal, and the remaining 75% are chosen by the learner itself. We don't know *which* of the actions are optimal, but we do know that 22.5% are.

Figure 5.12 represents the results for the 40x40 deterministic maze with all trainers. The x-axis represents the percentage of the trainer's actions that are optimal, the y-axis represents the percentage of the actions that are suboptimal, and the z-axis represents the mean number of actions. We wish to determine if there is a relationship between the number of optimal trainer's actions and the number of suboptimal trainer's actions. It is difficult to see in this graph, but it appears that given a particular percentage of optimal actions, increasing the number of suboptimal actions increases the total number of actions required to find an appropriate policy.

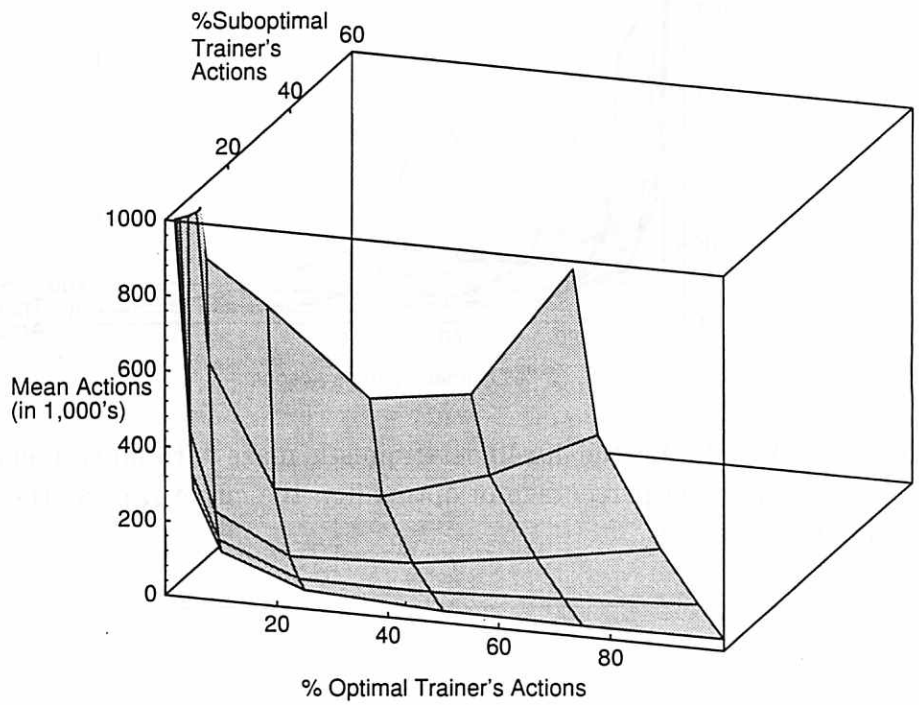



**Figure 5.12** Results for the 40x40 deterministic maze with all trainers, with respect to the percentage of optimal and suboptimal trainer's actions. The data is depicted in terms of the percentage of optimal trainer's actions versus the percentage of suboptimal actions.

This can be seen more easily in Figure 5.13, which views the three-dimensional data almost directly from the front. One can notice that as the amount of suboptimality increases (looking directly *into* the page), the curve tends to rise. A similar

view from the side indicates that for any given level of suboptimality, increasing the percentage of optimal actions decreases the amount of training necessary. Neither of these two findings is too surprising because one would expect that more good information will lead to improvement and more bad information will lead to degradation. A set of experiments aimed specifically at this question will provide better answers than this informal analysis.

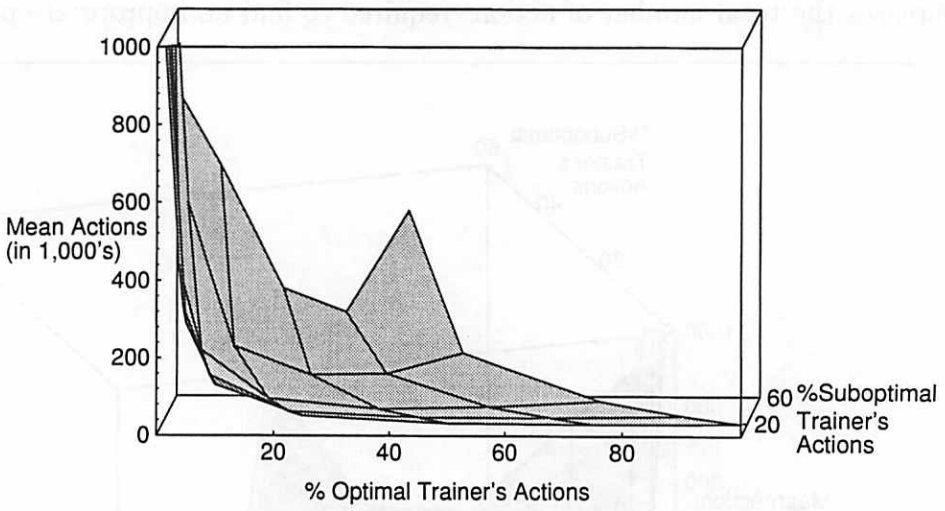



**Figure 5.13** Results for the 40x40 deterministic maze with all trainers, viewed from the front. For a given percentage of optimality, the curve rises as the percentage of suboptimality increases.

## 5.4 Scaling Up

For this experiment, we test how well the ASK FOR HELP approach with the uniform asking strategy scales to a more complicated task. To ask this question, we leave the comfortable maze world, with its simple lookup tables, and employ the race track task, which requires a generalization structure to represent its policy. Furthermore, because we cannot build an optimal policy for this task we do not have

an optimal trainer. Although the trainer can perform the task, we do not know how well it compares to optimal (see Section 5.1.2).

The results of the experiment are quite similar to the maze results: as the asking rate increases, the learner develops an appropriate policy more quickly. However, the results also show that asking rates lower than 100% are better than the 100% asking rate. These results are similar to those found in the previous experiments, lending credence to the conclusions we have drawn.

### 5.4.1 Experimental Design

In this experiment the task is the race track task. The task and the trainer are both described in Section 5.1.2. In the sub-experiments, we vary the learner's asking rate parameter from 0%, which is standard $Q$-learning, up to 100%, which is a form of apprentice learning. The learners attempt to drive on each of the two different race tracks. One is shown in Figure 5.3 and the other is an extension of that track that has additional turns and requires approximately twice as many actions to navigate properly.

Each individual sub-experiment consists of twenty runs, each of which begins with the $Q$-function weights set to zeroes. Because we cannot determine the optimal policy, the stopping criterion is based on a different performance measure. A run ends when the learner can navigate the car across the finish line from each of the eighteen designated starting positions on the track.

Each run consists of several trials, which all begin with the car placed randomly at one of the start positions. A trial ends with failure when the car drives off the track, after which a new trial begins. A trial ends with success when the car crosses the finish line, at which time the stopping criterion for a run is tested. If the stopping criterion is satisfied, the entire run ends. Otherwise, the run continues with the beginning of a new trial.

As in the maze experiments, for each run we record the number of actions necessary for the learner to achieve the stopping criterion and the number of trainer's actions. For the maze tasks we know that the policy developed is optimal. For this experiment we only know that the car can drive successfully from the start line to the finish via the learned policy.

Similarly to previous experiments, we perform statistical tests to determine whether results with non-zero asking rates are different than the results with $Q$-learning. We do not compare the results statistically to apprentice learning because the learners that asked the trainer 100% of the time do not learn appropriate policies in the allotted time.

### 5.4.2 Results

Figure 5.14 shows the results for the sub-experiments on the smaller track. The numerical data for this experiment are presented in the Appendix. As with the graphs for the maze task, this graph depicts the asking rate parameter on the independent axis and the mean number of actions necessary to achieve the stopping criterion on the dependent axis. The performance for standard $Q$-learning is shown with the dashed line labeled "Q-learning", with its 95% confidence interval marked by the dotted lines. The curve represents the number of actions required by the learner to meet the stopping criterion for different levels of asking rate.

Figure 5.14 indicates that AFH can also be effective for the race track task. At low asking rates, there is no significant difference between asking for help and standard $Q$-learning. Starting at the 20% asking rate and continuing up to 85%, asking for help is statistically better than not asking. The curve labeled "Learner" is quite similar to the curve in Figure 5.8 that represents a trainer that is incorrect 60% of the time. However, there is a major difference at higher rates of asking between the earlier results: the curve takes a steep rise starting at the 80% asking rate. The results for the 95% asking rate, which are off the graph, are significantly worse than

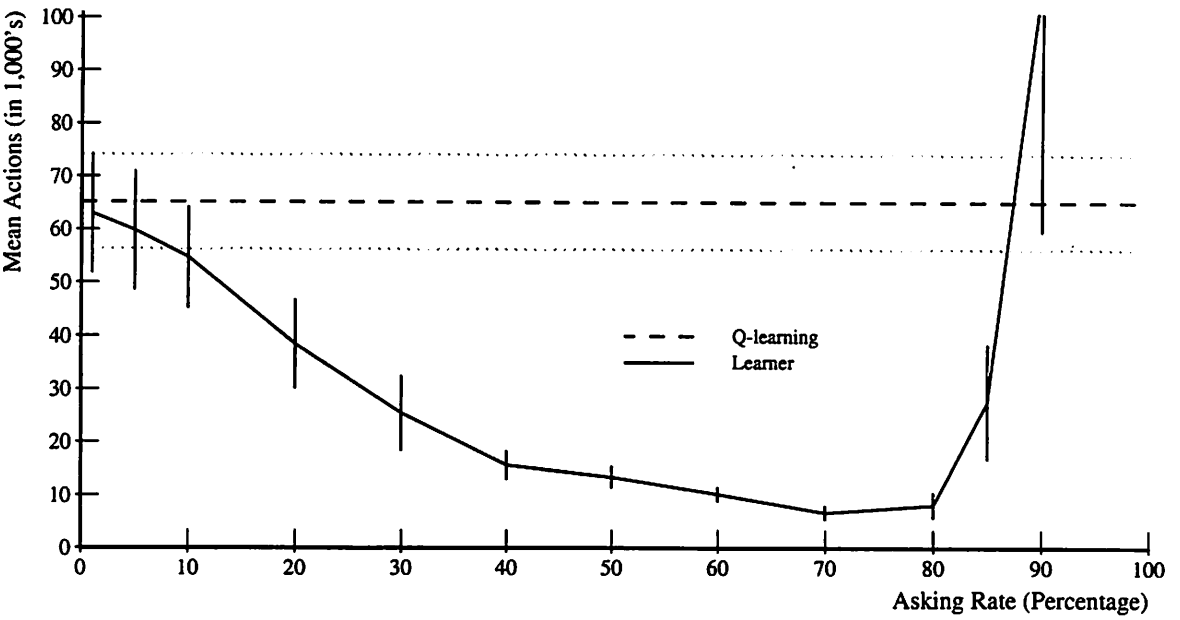



**Figure 5.14** Mean actions for race track task on track 1. The 95% confidence bars are shown. All means between the 20% and 85% asking rates are statistically better then those achieved at the 0% rate. The mean at the 95% asking rate is off of the graph.

not having a trainer at all. At the 100% asking rate the learner does not learn a policy for navigating the track: the runs were all stopped after the total number of trials exceeded 75,000. In comparison, $Q$-learning takes an average of less than 5,000 trials to complete.

The basic results for the longer track are the same as reported above for the shorter track, although the values of all the measured statistics are higher because the second track is longer. This data is shown in Figure 5.15. Note that the run at the 95% asking rate did not complete within the prespecified limits, and that the learner performed best at the 80% asking rate.

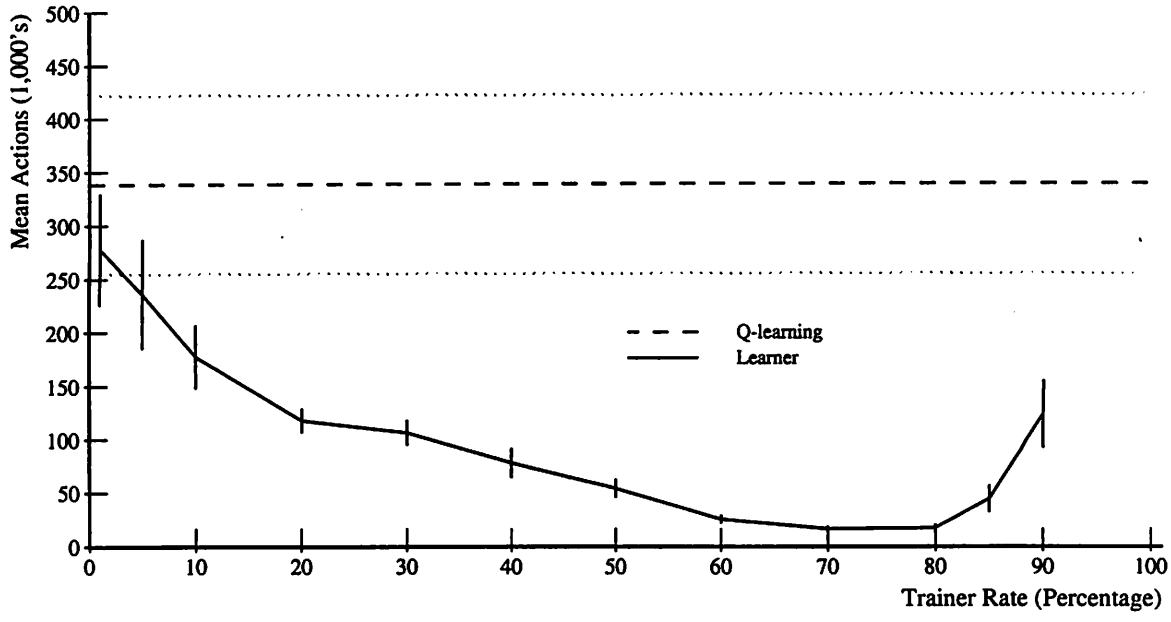



**Figure 5.15** Mean actions for race track task on track 2. The means between the 10% and 90% asking rates are statistically better than that achieved at the 0% rate. The runs with asking rates higher than 90% did not complete within the preset limits.

### 5.4.3 Discussion

Because the learner is able to satisfy the stopping criterion much more quickly with the trainer's aid than it does without the trainer's aid, we draw the same conclusion from this experiment that we do from the maze experiments: AFH is an effective means of integrating apprentice learning and reinforcement learning, at least at moderate levels of asking rate. However, these results are confounded by the use of linear networks to store the $Q$-values. Even so, the results are similar to those for the maze experiments where a trainer with a 60% error rate provides actions to the learner.

The results for this experiment show the tradeoff between apprentice learning and reinforcement learning nicely. At the far left of the graph in Figure 5.14 is standard $Q$-learning, and at the far right is apprentice learning with a suboptimal trainer.

Both of these methods require more training to achieve the stopping criterion than a combination of the two. As the asking factor increases from 0%, the amount of reinforcement learning decreases and that of apprentice learning increases. Clearly, a combination of the two is better than each individually, assuming the trainer is suboptimal.

The results of this experiment also indicate that the learner achieves the appropriate policy the quickest at either the 70% or 80% asking rate. This is quite similar to the results of the previous experiment that indicated that the 75% rate gave the best performance for a wide variety of error rates and maze sizes. This suggests that when applying AFH with the uniform asking strategy, one may wish to use an asking rate somewhere between 70% and 80%. Further experiments would need to be run in order to determine the exact rate.

The similarity of these results to those for the 60% error rate on the maze tasks suggests a possible technique for determining an agent's level of expertise. First, one would use the agent as a trainer, like we have in these experiments, and record the learner's policy acquisition rate for different asking rates. Then, to determine the trainer's expertise, one would find the error rate whose data matched the agent's data most closely. Although we do not know that this technique will be effective, the observation that the curves for this experiment and the previous experiments are similar suggests this as an avenue for future research.

## 5.5 Asking Based on Uncertainty

For this final experiment, we compare the two asking strategies, uniform asking and uncertainty asking. We wish to determine whether a more sophisticated approach to querying the trainer is better than the random approach, which will give us insight into when it is appropriate to receive training information. We find that the

uncertainty approach is better, allowing the learner to acquire an optimal policy with fewer total actions while receiving the same amount of trainer support.

### 5.5.1 Experimental Design

Like the first two experiments, the domain for this experiment is the maze domain. For the first two experiments (Section 5.2 and Section 5.3), the learner queries the trainer based on an asking rate parameter, which determines the percentage of the time that the learner asks for help. In this experiment, the learner bases the decision of when to ask on the status of its current policy. When deciding on an action to choose, the learner examines the highest and lowest $Q$-values. If the difference between these values is larger than the setting for the asking factor, the learner will ask the trainer for an action. In each of the sub-experiments, we vary that asking factor, changing it from 0.0, which requires that all of the current move choices have equal values in order for the learner to ask for help, up to 1.0, a setting that causes the learner to ask for aid all of the time.

Each individual sub-experiment is performed in the same manner as in the first two experiments. There are ten runs, each of which begins with zeroed $Q$-values and ends when either the learner meets the stopping criterion or surpasses the preset action or trial limits. Each run consists of several trials in which the learner attempts to navigate from the start cell to the goal.

### 5.5.2 Results

We first show the results of this experiment in a graph similar to those in which we presented previous maze results. In the next section, we display the results in a different type of graph in order to facilitate comparison with results from the experiments with the uniform asking strategy. The numerical data for this experiment are presented in the Appendix. Figure 5.16 shows the results for the 40x40 maze. Like previous graphs, the dependent axis shows the mean number of trials. The inde-

pendent axis shows the asking factor. Each curve represents the results for different levels of trainer error.

For the 40x40 deterministic maze, standard $Q$-learning requires an average of 1.8 million actions to find an optimal policy. The results for every sub-experiment are significantly better than the result for $Q$-learning. The highest mean number of actions observed, which is for the 50% error rate and 0.0 asking factor, is only 150,000. Thus, AFH with the uncertainty strategy is better than $Q$-learning alone. Furthermore, AFH performs better than apprentice learning alone. With an asking factor of 1.0, the learner ends up asking the trainer for aid 100% of the time, and so the sub-experiments at the 1.0 asking factor represent a form of apprentice learning. One can observe that each of the curves tends to rise as the asking factor increases, indicating that lower asking factors produce results better than those at the 1.0 factor. In fact, the mean number of actions required to meet the stopping criterion is never statistically worse than those required with apprentice learning.

For the smaller mazes, the results tend to be significant starting at the 0.0001 asking factor and continuing to the 0.75 factor, with few exceptions. For the large mazes, all results tend to be statistically better than those for apprentice learning. It is also the case for the stochastic mazes that the results at all asking factors and with all suboptimal trainers are better than, or at least indistinguishable from, the apprentice learning results. For the optimal trainers, the results are significantly worse at low asking factors, but indistinguishable at the higher factors. Thus, for suboptimal trainers, AFH with the uncertainty strategy is better than apprentice learning.

### 5.5.3 Discussion

We first conclude that the ASK FOR HELP approach with the uncertainty asking strategy is effective for integrating apprentice learning and reinforcement learning, even for suboptimal trainers. Further, with the uncertainty strategy, the learner can

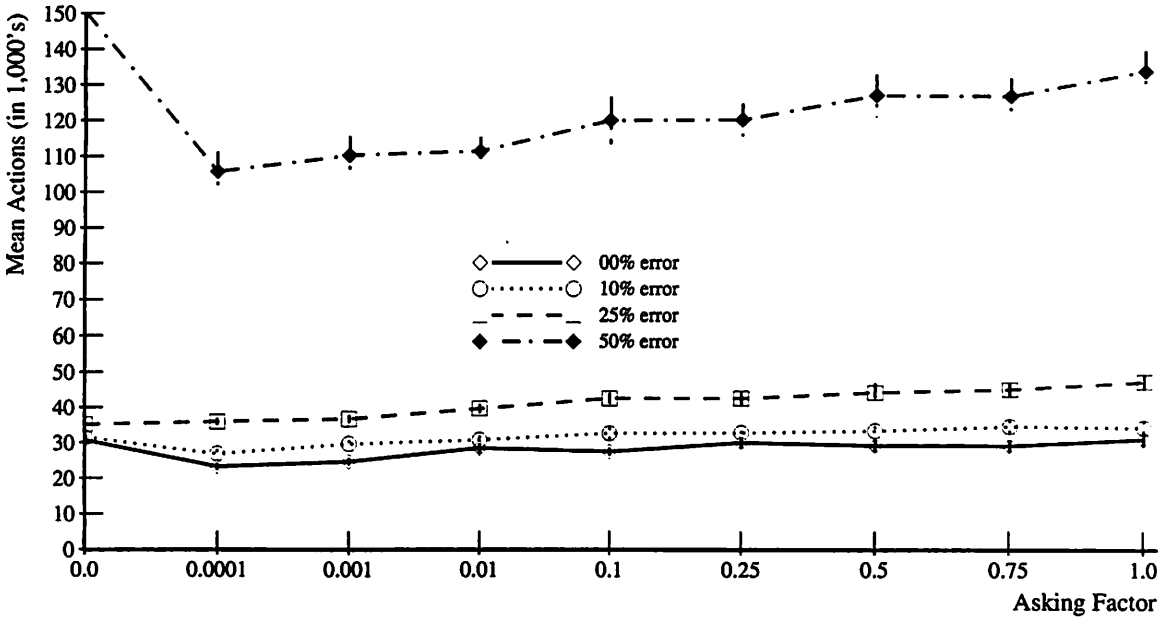



**Figure 5.16** Results for the 40x40 deterministic maze with all trainers and the uncertainty asking strategy. All of these means are statistically better than the $Q$-learning mean, which is off of the graph. With $Q$-learning the agent requires close to 6 million actions on average to find an optimal policy.

reduce its interaction with the trainer and still learn an optimal policy in as much time as though it had asked for aid continually. In the remainder of this section we compare the results of the experiment with the uniform asking strategy to those with the uncertainty asking strategy.

To facilitate comparison with the results based on the uniform asking strategy, we introduce a new type of graph. Before showing the results for this experiment in this new form, we present the results from the first experiment. Like previous graphs, Figure 5.17 shows the number of actions needed to meet the stopping criterion on the vertical axis. The horizontal axis is the number of trainer's actions that were received, not the asking rate. The data presented on this graph represents the same data as that in Figure 5.4 on Page 83.

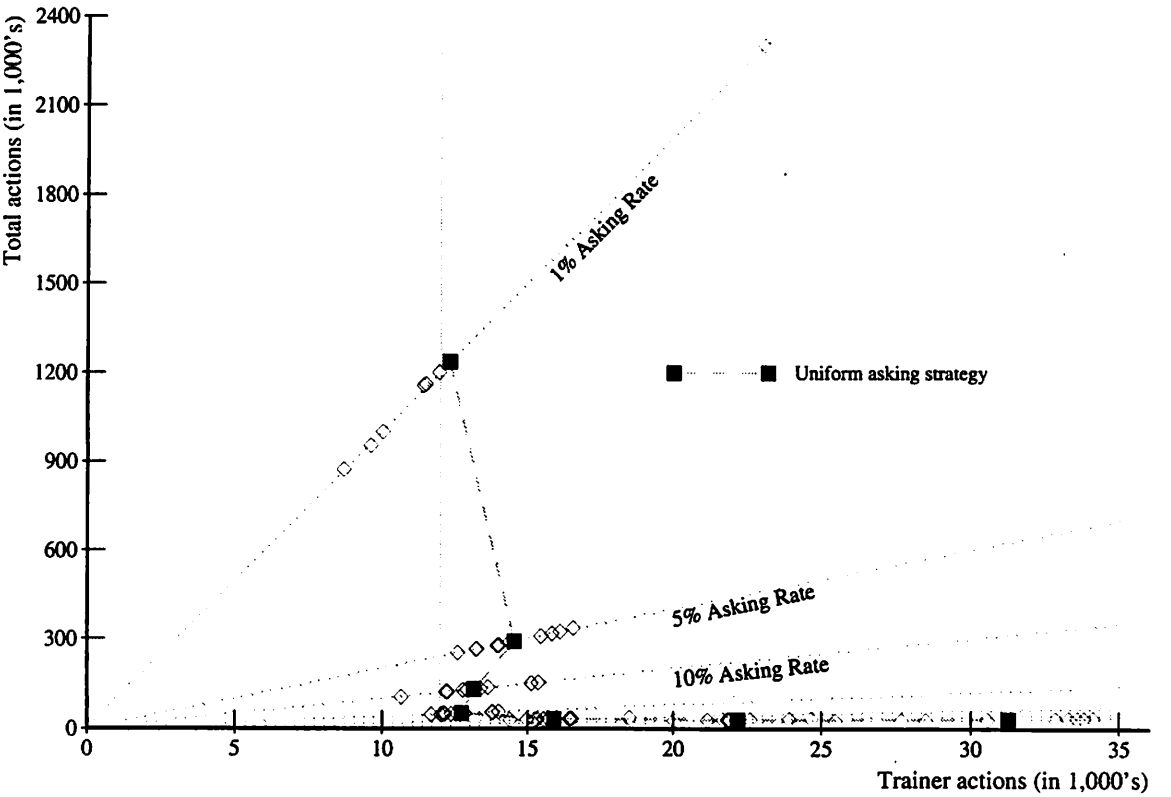



**Figure 5.17** Scatter graph for the 40x40 deterministic maze with the optimal trainer and the uniform asking strategy. These are the same results depicted in Figure 5.4 (page 83), but represented differently.

Each diamond on the graph represents the result of each individual run of each sub-experiment for the 40x40 deterministic maze with an optimal trainer. The dotted lines represent the different asking rates. For example, the diamonds that lie on the line labeled "1% Asking Rate" are the results from the sub-experiment in which the learner asked the trainer for an action 1% of the time. Because the rate is a controlled parameter, every run at the 1% asking rate will produce a result that will lie on this line. In order to avoid cluttering the graph, only three of the seven lines associated with the asking rate sub-experiments are labeled. The results for each of the seven asking rate sub-experiments has an associated center, which is at the coordinates of

the mean of the two statistics for that particular asking rate: the x coordinate is the mean of the number of trainer's actions, and the y coordinate is the mean of the total number of actions. This center is depicted as a dark square. The light, dashed line simply joins these center points.

The graph also depicts a light vertical line at 12,000 trainer's actions, which helps show that not all trainer's actions are equally useful in speeding up the learner's ability to acquire an optimal policy. For the same number of trainer's actions, the learner requires vastly different amounts of total training. This can be seen by traversing the vertical line from the x-axis upwards. With 12,000 actions the learner is able to learn the task in approximately 48,000 total actions (at the 25% asking rate), 120,000 actions (at the 10% asking rate), and 1,200,000 actions (at the 1% asking rate). This phenomenon is not particular to 12,000 trainer's actions, it can also be observed at other levels of trainer interaction. Because the same number of trainer's action produces these widely varying results, one can say that certain trainer's actions are better than others. For example, the 12,000 trainer's actions given at the 1% asking rate were not as effective as those received at the 25% asking rate. One can also view this finding from the viewpoint that the trainer's that respond 25% of the time are more effective than those that respond only 1% of the time.

Figure 5.18 represents the same information as Figure 5.17, but shows the results of a sub-experiment with the uncertainty asking strategy. The circles represent the measured statistics for each individual run, with a dark diamond in the center of the results for each asking factor. The dark diamond to the far left represents the data for the sub-experiment with the 0.0 asking factor, the next diamond is at the 0.0001 asking factor, and the last is at the 1.0 asking factor. Unlike the results from the first experiment, notice that these groupings do not generally lie on a line through the origin. This is especially true on the left of the graph. However, the last group of results on the far right of the graph are on the line corresponding to 100% interaction

with the trainer. Also notice that the vertical axis has a scale different from the previous figure.

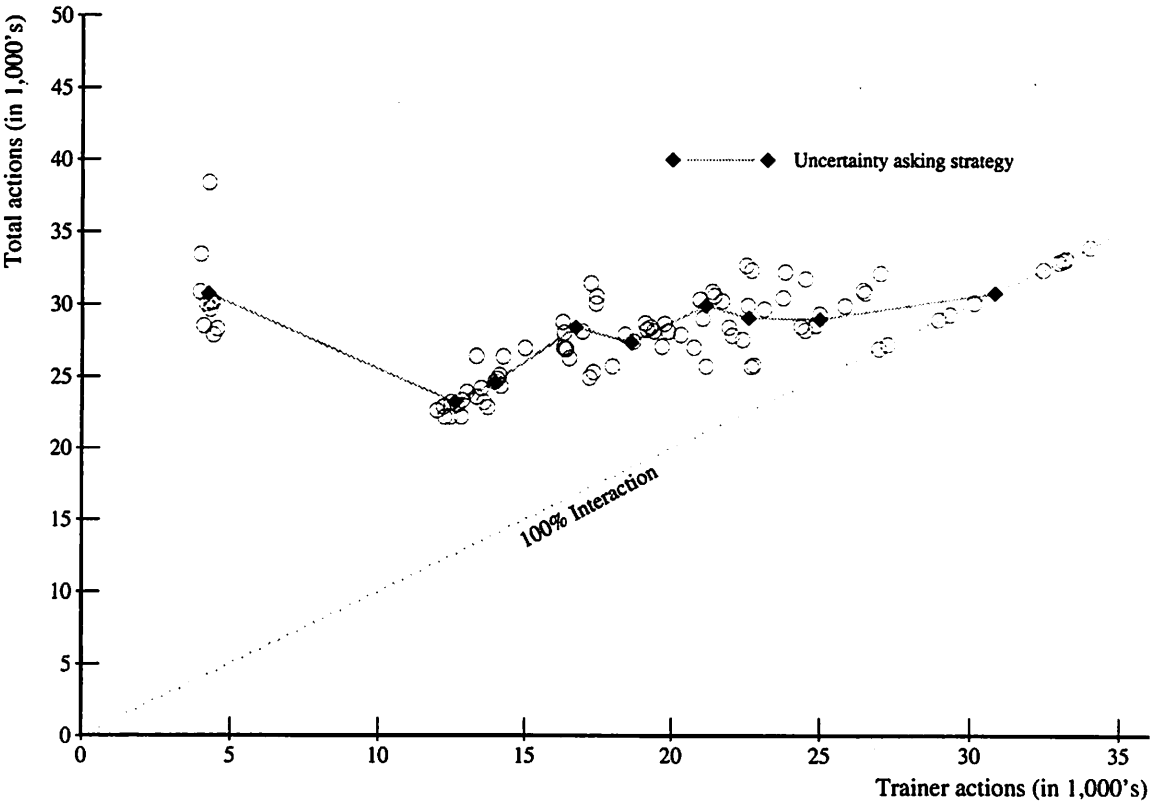



**Figure 5.18** Scatter graph for the 40x40 deterministic maze with the optimal trainer and the difference asking strategy. These results are the same as those for the optimal trainer in Figure 5.16 (page 105).

In order to compare the two sets of results, Figure 5.19 depicts them on the same graph. In general, the points plotted for the uncertainty strategy experiment are below those of the uniform strategy experiment. This is most noticeable in the group of points near 4,000 trainer's actions. With approximately 4,000 trainer's actions, the learner develops an appropriate policy in only 30,000 actions. With the uniform asking strategy, the learner needs close to 15,000 trainer's actions in order to achieve this level of performance. Thus, with the uncertainty strategy, the learner needs fewer

trainer's actions to reach the same level of performance as with the uniform strategy. Also notice that in this particular case, the proportion of trainer's actions in the total number of training actions is close to 15%.

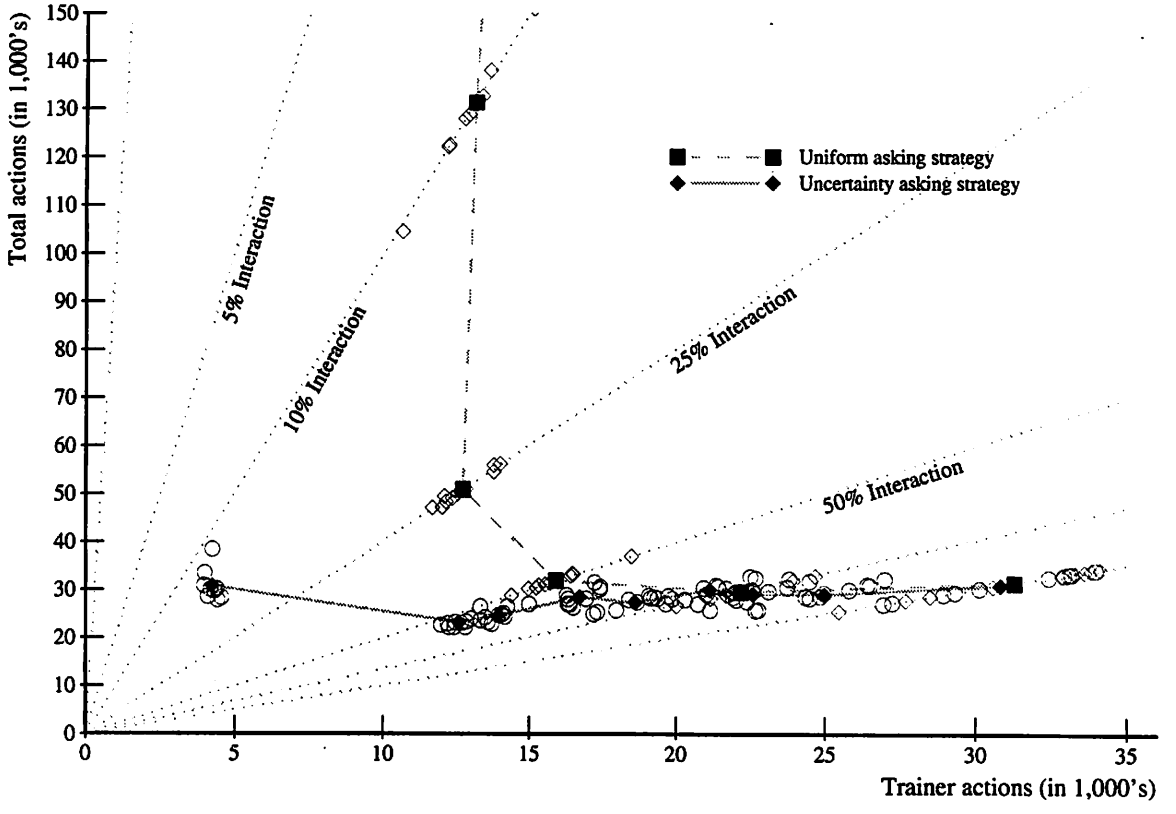



**Figure 5.19** Scatter graph for the 40x40 deterministic maze showing both asking strategies with the optimal trainer. Note that the curve for the uncertainty asking strategy never higher than the curve for the uniform asking strategy, indicating that the uncertainty asking strategy produces better results.

One can also compare how well the learner performs for the same amount of trainer interaction but with different asking strategies. For example, with the uncertainty strategy and 12,000 trainer's actions, the learner finds an optimal policy in approximately 23,000 total actions. With only 12,000 trainer's actions, the learner using the uniform asking strategy requires at least 48,000 total actions to meet the stopping

criterion. Thus, the uncertainty strategy makes better use of the trainer, achieving quicker convergence with the same number of trainer's actions.

With the uniform strategy, as the name suggests, the trainer interacts with the learner uniformly throughout training. With the uncertainty strategy, the amount of interaction with the trainer diminishes over time during a training run. This is depicted in Figure 5.20, which plots the number of trainer's and learner's actions for each trial of a particular run with the 40x40 maze, the optimal trainer, and the 0.0 asking factor. The horizontal axis in the graph shows the number of actions that the learner chooses on its own, and the vertical axis represents the number of actions chosen by the trainer. Each point on the graph represents the number of learner's and trainer's actions for a particular trial. For example, the point at (48, 36), which is marked with the dotted lines, represents a trial in which the learner asked the trainer for an action 48 times, and performed its own action 36 times, for a total of 84 actions.

For the first trial, the learner does not choose any of its own actions, and the trainer performs the task optimally. The point for this is plotted at (0, 74), signifying that the trainer provided every action in performing the task. As training progresses, the number of trainer's actions tends to decrease while the number of learner's actions increases. Toward the end of the run, very few of the actions come from the trainer, which is shown by points near the horizontal axis. At the end of the run, the learner performs 74 actions, the optimal number. These results, which are similar for other runs, other mazes, and other trainers, indicate that the learner relies heavily on the trainer at the beginning of training, but asks for help less often as training progresses. This is a positive result when one considers having a human as a trainer. Instead of needing to provide actions a uniform percentage of the time throughout training, the human will provide many actions at the beginning of training and be needed less and less as training progresses.

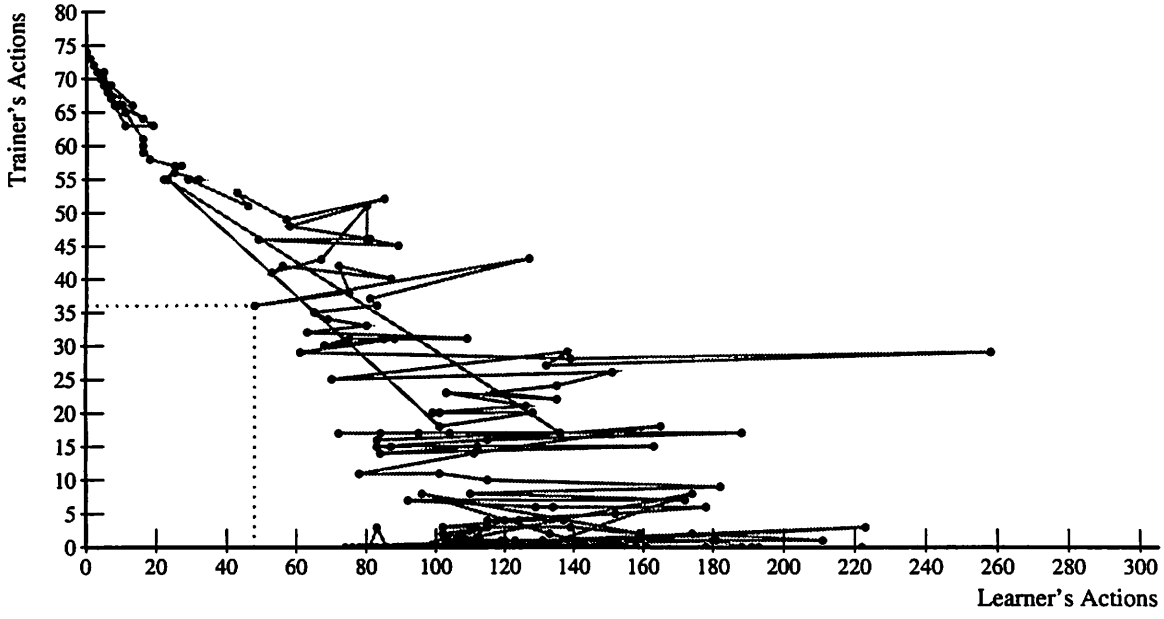



**Figure 5.20** Results for the 40x40 maze showing the trainer's action distribution. The data for the early trials are along the dependent axis, and the data for the trials near the end of the run are along the independent axis. The line segments connecting the points show the progression from the first trial to the last.

Like the uniform asking strategy, with the uncertainty strategy the learner visits only a small portion of the state space. For example, Figure 5.21 depicts the frequency of cell visits for the 40x40 maze on a particular run at the 0.0 asking factor with an optimal trainer. The height of each bar in the graph depicts how often the corresponding maze cell was visited. The start cell is at the left corner of the graph, and the goal cell is in the right corner. Almost all of the learner's experience was acquired in a particular area of the graph (depicted with the highest bars). For this particular run, the learner visited only 255 of the 1,018 cells, a reduction by a factor of four.

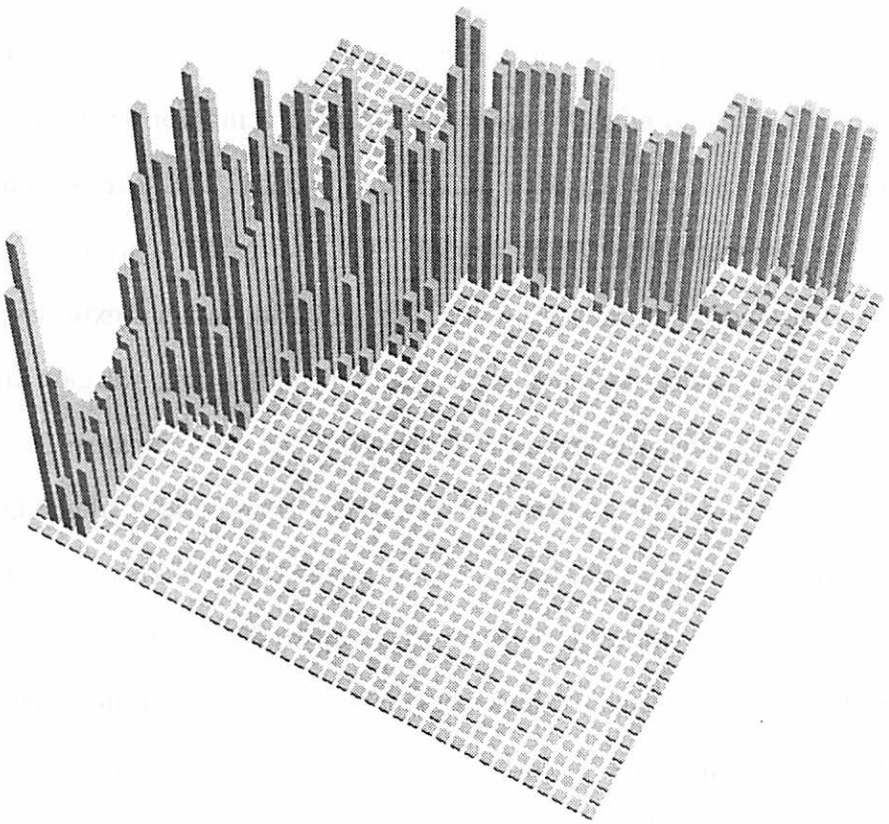


**Figure 5.21** Frequency of cell visitation for the 40x40 deterministic maze with the uncertainty asking strategy on a run in which the learner asked the optimal trainer with an asking factor of 0.0.

## 5.6 Summary

The main objective of the experiments is to answer the general questions presented in Chapter 4: how should the learner and trainer interact, how should the learner incorporate the trainer's actions into its developing policy, how does the trainer's expertise influence the learner's ability to acquire an appropriate policy, and when should the trainer provide information to the learner? Secondarily, we also wish to determine whether ASK FOR HELP is an effective approach to integrating apprentice learning and reinforcement learning.

The results of the first experiment indicate that the trainer's aid does speed up the learning process, allowing the learner to develop an appropriate policy significantly faster than when it relies on standard $Q$-learning. Furthermore, these results show that AFH is an effective approach for integrating apprentice learning and reinforcement learning, at least with an optimal trainer. The learner is able to learn an optimal policy while also taking advantage of the trainer's proffered actions

In the second experiment, in which we vary the trainer's expertise by controlling how often the trainer responds with a suboptimal action, we gain further evidence that AFH is an effective means of integrating the two disparate learning methods. Moreover, we find that the trainer's expertise does indeed affect how quickly the learner produces an appropriate policy. Although this conclusion is not too surprising, we also find that a trainer that is incorrect up to 25% of the time helps the learner almost as much as a perfect trainer. AFH is useful even when the trainer is not perfect. This finding is important when having humans serve as the trainers, allowing those that do not perform the task optimally to be considered.

The results from the third experiment indicate the the AFH approach also works for more complex tasks than maze navigation. The learners here rely on a hand-coded trainer, and learn more quickly with that trainer under certain conditions. Even though the race track task is different from maze navigation, the results of the experiments are quite similar to those for the maze tasks. The experiment shows the tradeoff between apprentice learning and reinforcement learning, strengthening the hypothesis that an integrated method is better than its constituents.

In the fourth experiment we compare the uniform asking strategy to the uncertainty asking strategy. We find that the more principled strategy allows the learner to benefit from the trainer better, requiring fewer trainer's actions to achieve the same level of performance. Thus, the learner can employ a more intelligent strategy for when to ask the trainer for aid, and reduce its reliance on another agent. This is par-

ticularly important when there is a cost associated with each of the trainer's actions. Moreover, every trainer's action does not provide the same amount of information to the learner in aiding the learner to develop an optimal policy quickly.

Evidence from the maze tasks suggests that the ASK FOR HELP approach possesses the desirable characteristics described in Chapter 3 (Section 3.1.1). It allows the trainer to learn from reward signals as well as from the trainer's actions. It exposes the learner to much less of the space than when it relies on $Q$-learning alone. And, finally, the policies learned are optimal. An approach of simply asking for actions uniformly throughout training, and simply performing the action when it is received, is an effective method for learning to perform multiple-step tasks.

We demonstrate that ASK FOR HELP allows the learning agent to produce correct policies more quickly than with either reinforcement learning alone or apprentice learning alone. We also show that the uncertainty asking strategy performs better than asking for aid randomly. The results show that the expertise of the trainer takes a major role in the success of the learner. Finally, we conclude that one can indeed integrate apprentice learning and reinforcement learning to advantage.

# CHAPTER 6

# DISCUSSION

This dissertation is about integrating apprentice learning and reinforcement learning, and is focused on answering particular questions about the integration of these two learning methods. In addition to finding the answers, the research has raised a set of issues for consideration in the future. Moreover, in developing, implementing, and studying a particular integrated approach, the dissertation has contributed towards understanding better the integration of apprentice learning and reinforcement learning.

## 6.1 Summary of the Dissertation

Previous research in integrating apprentice learning and reinforcement learning has identified the integration of the two methods as promising for building automated agents that can learn to perform multiple-step tasks. This dissertation furthers the understanding of hybrid systems by examining particular aspects of the integration not addressed by previous work.

A thorough comparison of the two individual methods reveals that each possesses strengths as well as weaknesses and that neither is clearly better than the other. In Chapter 3, we argue that the two methods have complimentary strengths and that the hybrid of the two will be a more powerful method than either individual. In considering an integrated method, we develop the novel learner/trainer model, which makes explicit many of the possible interactions that can take place between the learner and the trainer, and introduces the notion of an interaction policy—a policy

that specifies how the learner and trainer will interact in the hybrid method. Our main objective is to answer particular questions about integrated methods. We designed our ASK FOR HELP approach with these questions in mind. We wish to determine: how the learner and trainer should interact, how the learner should incorporate the trainer's actions into its developing policy, how the trainer's expertise influences the learner's ability to acquire an appropriate policy, and when it is effective for the trainer to provide information to the learner.

In the remainder of this section we first summarize the ASK FOR HELP approach. We then present the empirical results that confirm the hypotheses.

### 6.1.1 The ASK FOR HELP Approach

In order to study certain aspects of integrating apprentice learning and reinforcement learning, we developed a new integrated approach, ASK FOR HELP. Although previous research has shown some benefit in integrating these two learning methods, that research has left several questions unanswered. Our objective in developing AFH is to tackle four of those questions.

In the ASK FOR HELP approach, the learning agent employs $Q$-learning as its learning algorithm. This gives the learner the ability to develop an optimal policy based on scalar reward signals. In addition to learning from reward signals, the learner also has access to actions provided by a training agent. The learner incorporates a trainer's action into its policy simply by acting as though it had chosen the action itself: perform the action, receive whatever reward is provided, and update the $Q$-functions accordingly. Thus, to incorporate an action requires no changes to $Q$-learning.

The learner is completely responsible for its interaction with the trainer, asking the trainer to provide actions. The trainer will always give an action when asked. The learner employs one of two strategies to ask for help. With the first strategy, the learner asks the trainer to provide actions randomly throughout training. The

motivation for employing such a simple strategy is to determine when it is effective to receive the trainer's advice in order to help the learner develop a policy more quickly than not having access to a trainer. The second asking strategy relies on the $Q$-values of the learner's current action choices to determine whether to ask for aid. When the difference between the highest and lowest $Q$-values is within a pre-specified limit, the learner will ask for help. This strategy allows us to ask whether a principled approach to asking for aid is an improvement over the first strategy.

### 6.1.2 Empirical Study

The empirical study demonstrates that ASK FOR HELP is an effective approach to integrating apprentice learning and reinforcement learning. In each of the experiments, the learning agent develops its policy more quickly than with reinforcement learning alone. Furthermore, the results indicate that AFH sometimes allows the learner to develop a policy more quickly than it can with only apprentice learning. The experiments also indicate each of the following: actions received at random times in the training are useful, the trainer's expertise influences how quickly the learner develops an optimal policy, the AFH approach can scale up to a larger domain than maze tasks, and the more sophisticated asking strategy allows the learner to learn even more quickly.

The results of the first two experiments indicate that the trainer's aid does speed up the learning process, allowing the learner to develop an appropriate policy significantly faster than when it relies on standard $Q$-learning alone. This occurs partially because the trainer focuses the learner on parts of the state-space that are appropriate for learning the optimal policy. Recordings of the actual maze cells that were visited indicate that the learner is exposed to cells within a neighborhood of the optimal path; the size of the neighborhood is determined loosely by the asking rate and by the expertise of the trainer. Roughly speaking, the higher the asking rate and the higher the expertise, the smaller the neighborhood and the faster the learning.

Additionally, the second experiment demonstrates that the trainer's expertise does indeed influence the rate at which the learner develops an optimal policy. Because the trainers are stochastic degradations of an optimal trainer, these results suggest that other suboptimal trainers may also be useful. If these degraded trainers are not able to help the learner, the efficacy of other trainers would be seriously in doubt.

Given that the ASK FOR HELP approach is effective for the maze tasks and that the suboptimal trainers are helpful, the next experiment is designed to determine whether AFH will scale to a larger task with a suboptimal trainer (one that is not a simple degradation of an optimal trainer). With most settings of the asking rate, the learner develops a policy significantly more quickly than without the trainer's aid. The results of this experiment also show that the integrated method produces a policy that meets the stopping criterion when a form of apprentice learning with a suboptimal trainer does not.

In the final experiment, the learner is able to learn an optimal policy more quickly with the uncertainty asking strategy than with the uniform asking strategy, while receiving the same number of trainer's actions. This indicates that the more sophisticated strategy identifies places where the trainer's actions are more informative than trainer's actions received randomly. This experiment further supports the claim that an integrated method can be better than apprentice learning: as the ratio of trainer's to learner's actions approached one (getting closer to apprentice learning), the total number of actions necessary to achieve the stopping criterion increases given a suboptimal trainer.

The empirical study indicates that an integration of apprentice learning and reinforcement learning can be better than its constituents, and that the ASK FOR HELP approach is effective for integrating these two learning methods. Furthermore, the results suggest that humans can serve effectively as trainers. Even when the human is not optimal, the learner can benefit from the interactions that take place. Fur-

thermore, the uncertainty asking strategy will make good use of the human's time, requiring less interaction than a random approach, and requiring less interaction as training progresses. Thus, the AFH approach is applicable to situations in which a human serves as a training agent.

## 6.2 Issues for Future Research

This dissertation exposes many avenues for future research. Chapter 3 raised several issues that must be addressed when building an integrated system. In considering these issues, we made certain decisions when designing AFH, for example insisting that the trainer provide actions to the learner. In this section we discuss other options for dealing with those issues, reexamining the form of advice provided by the trainer, the mechanism for incorporating that advice into the policy, and the interactions that take place between the learner and trainer.

### 6.2.1 Form of Advice

The ASK FOR HELP approach relies on advice from the trainer in the form of actions to perform. There are at least two other types of advice that the trainer might provide. The trainer could give advice as scalar signals (similar to the rewards received) that criticize the learner's most recent actions. As noted in Section 3.4.2, this form of advice has been used previously with success, indicating that the trainer's evaluative feedback, in addition to the rewards, might be beneficial to the learner. Like actions, this form of advice, is a natural and easy means to convey information to a learning agent. The trainer does not even need to know which actions to perform, but can simply note whether the learner's actions are good or bad. Because there are many questions to answer before this form of advice becomes used widely, this area of research should be explored further.

The approach taken in AFH is a special case of another possible form of advice. Instead of giving only one action that is suitable to perform, the trainer might give the learner an entire set of actions from which to choose one. Preliminary work in the race track domain indicates that this form of advice also improves the learner's ability to solve the problem quickly (Clouse, 1995). Furthermore, the trainer in that study cannot solve the problem itself; it can only identify actions that are clearly bad and should not be placed in the set of actions, and those that are clearly good and should be placed in the set. The learner benefits because the trainer has already narrowed the learner's options, allowing the learner to concentrate on fewer action choices. This type of advice may also keep the learning agent from making costly mistakes as it trains because the learning agent is constrained to choosing from a subset of the possible actions.

### 6.2.2 Incorporating Advice

Regardless of the form of trainer's advice, the advice must be incorporated into the learner's policy. In changing the learner's policy based on the trainer's advice, care must be taken to ensure that the learner can still develop an optimal solution. For this reason, the incorporation mechanism of AFH has the learner simply perform the trainer's action. Several other options exist. For example, instead of performing the action, the learner might raise the $Q$-value of the current state and trainer's action slightly, and then choose whatever action has the highest $Q$-value (it may be the trainer's action, but it may not). In addition to performing the trainer's action, the trainer might also increase its associated $Q$-value regardless of the reward received. Clouse & Utgoff (1992) took a similar approach in their early integrated system. Many possible options exist that have not been explored fully.

Incorporating scalar advice can be as simple as treating the advice as a reward signal. This would not require any changes to the underlying reinforcement learning algorithm because it was designed to update based on scalar reward signals. At least

two difficulties arise when considering criticism as a form of advice. First, one must decide how to handle the conflict between the reward signal and the trainer's signal. Options for dealing with this include ignoring the trainer, ignoring the reward signal, or combining the two mathematically. This area has not been explored much. The other difficulty is related to the trainer's expertise at the problem. As with other forms of advice, one must determine how proficient the trainer needs to be in order to be helpful. One should be especially careful that the learner maintains its ability to learn an optimization of the reward signals—its ultimate objective—even though it is also incorporating scalar signals from the trainer.

As with the previous two forms of advice, there are at least two options for incorporating advice in the form of sets of actions. First, the learner simply chooses the action it considers to be the best of the set and performs it. Once an action is chosen, the options for incorporating it are the same as if the trainer had given only one action, as discussed above. In the second option, the learner performs whatever action it chooses, and updates its policy depending on whether the action it chose was in the set or not. If the action was in the set, the learner increases the action's $Q$-value regardless of the reward received, thus "rewarding" the learner for picking one of the trainer's actions. Conversely, if the learner picked an action that was not in the trainer's set, the learner gets "punished" for doing so, decreasing the chosen action's $Q$-value. Of these two assimilation options, the first seems the least prone to difficulties. As with other incorporation schemes, one must be careful that the learner still has the ability to optimize the receipt of the rewards received.

### 6.2.3 Learner/Trainer Interactions

The ASK FOR HELP approach, like all previous integrated systems, employs particular policies that specify how the learner and trainer interact. As discussed in Section 3.2, the choices made are not the only possibilities. The learner and trainer may be involved in more sophisticated interactions than any system to date has con-

sidered. For example the learner and trainer could engage in a dialog about the current problem, similarly to PROTOS (Section 2.2.2).

For the empirical study, we employed the "Ask for Help?" interaction policy and implemented it via two strategies. The first strategy is straightforward, but the second is more sophisticated. The later strategy, which we call the uncertainty asking strategy, relies on the difference between the agent's $Q$-values. In the experiments we examine how a change in the asking factor, which determines whether the learner is uncertain, affects the learner's acquisition of an appropriate policy. Unfortunately, the setting of the asking factor depends greatly on the magnitude of the $Q$-values. For our problem, the $Q$-values remain less than 1.0, and so we use asking factors less than 1.0. However, for other problems, the $Q$-valuesmay get quite large, requiring larger asking factors to ascertain whether the learner is uncertain of its current choice. It is an issue for future research to determine how to set the asking factor. However, our empirical study indicates that setting the factor at its lowest level, 0.0, is affective. This level of the asking factor is applicable to any problem, because it indicates that all the current actions choices have identical values.

The learner and trainer might also *learn* the interaction policies. After all, the interaction policies are just policies: given a current state they produce an action. For an interaction policy, the state may include information about the learner and the trainer as well as the task state. It may be possible to develop these policies with reinforcement learning, apprentice learning, or even an integrated method. If the learner were learning the "Ask for Help?" interaction policy, it might learn to ask for help frequently at the beginning of training and less often as training progresses. Or, the learner might learn to recognize a bad trainer and to quit asking for help. After the learner or trainer has learned a particular interaction policy, it may apply that policy to learn from or train another agent on a similar, or possibly quite different, problem. This is a rich area for future research.

## 6.3 Conclusion

As computer systems become more complex, and the tasks we humans expect them to perform become more demanding, programming computers directly will become increasingly impractical. Consequently, machine learning will take on a more pervasive and essential role in building computer systems. As a step in this direction, this dissertation focuses on automated agents that learn to make a sequence of decisions in order to solve a problem. Taking inspiration from human learners, who not only learn from others but also refine their skills autonomously, this dissertation has concentrated on issues that arise in integrating apprentice learning and reinforcement learning. Although prior research has explored this area, that research has not focused on the specific issues addressed herein.

This dissertation shows that a careful integration of the two seemingly disparate learning methods can produce a more powerful method than either one alone. First, an argument based on the characteristics of the two learning methods maintains that a hybrid method will be an improvement because of the complimentary strengths of its constituents. Second, the empirical study shows that the ASK FOR HELP approach integrates the two methods effectively. The study demonstrates that even the straightforward technique of asking the trainer for actions randomly throughout training and then incorporating those actions into the developing policy by simply performing them is clearly better than reinforcement learning alone, and sometimes better than apprentice learning alone. Moreover, the results indicate that the trainer's expertise in performing the task must be considered thoroughly, because it has a direct bearing on how well the learner will benefit from the trainer's suggested actions, and may even preclude the learner from acquiring an appropriate policy. Thus, one must be careful when designing an integrated system to examine the trainer's ability to perform the task. Finally, the uncertainty asking strategy employed in this research

is an improvement over asking for aid uniformly. This suggests that sophisticated strategies for obtaining information from the trainer are worth studying further.

With continued research, we believe that expansion on the work presented in this dissertation will lead to other techniques that can be applied to tasks that are not currently amenable to a learning approach—tasks for which current techniques require prohibitive amounts of time or space to learn, and tasks that are not well enough understood to program directly. Eventually, computers will no longer need to be programmed directly to perform complex problem-solving and control tasks; instead, they will be taught to perform these tasks, through the integration of apprentice learning and reinforcement learning.

# APPENDIX

# DATA FOR ALL THE EXPERIMENTS

The data provided in this appendix are the results of the experiments described in Chapter 5.

The following tables give the data for the maze experiment with the optimal trainer, which is discussed in Section 5.2 starting on page 79. In all of the tables, the column labeled "Actions" shows the mean and 95% confidence interval for the number of actions needed to achieve the stopping criterion, in 1,000's. "Trials" shows the mean and confidence interval for the number of trials performed.

| Data for the 20x20 deterministic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 135.81±38.34 | 608.78±730.64 |
| 1 | 128.78±64.68 | 1174.50±1538.13 |
| 5 | 41.57±4.54 | 146.00±10.78 |
| 10 | 23.56±1.72 | 143.50±10.27 |
| 25 | 12.16±0.54 | 151.70±7.62 |
| 50 | 7.97±0.24 | 157.50±5.46 |
| 75 | 6.66±0.23 | 164.60±6.27 |
| 100 | 6.61±0.25 | 183.50±7.05 |

| Data for the 40x40 deterministic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 1800.69±561.96 | 3667.56±7206.77 |
| 1 | 1235.99±449.95 | 2754.71±5364.82 |
| 5 | 290.44±20.30 | 313.50±24.03 |
| 10 | 131.21±10.14 | 297.90±18.56 |
| 25 | 50.86±2.47 | 313.60±18.47 |
| 50 | 31.86±1.65 | 315.70±18.81 |
| 75 | 29.56±1.25 | 359.70±16.05 |
| 100 | 31.27±2.19 | 422.60±29.55 |

| Data for the 60x60 deterministic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 5963.28±1525.81 | 7104.30±9407.78 |
| 1 | 6473.80±4842.51 | 26444.20±35343.46 |
| 5 | 1021.48±70.98 | 774.10±53.69 |
| 10 | 492.58±29.68 | 727.70±36.62 |
| 25 | 180.51±6.01 | 681.00±20.88 |
| 50 | 120.82±1.49 | 768.50±9.92 |
| 75 | 125.65±2.75 | 984.90±21.20 |
| 100 | 154.19±2.99 | 1352.50±26.22 |

| Data for the 80x80 deterministic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 29745.13±9581.29 | 47278.86±46996.19 |
| 1 | 10222.91±601.03 | 1498.10±938.10 |
| 5 | 2303.58±86.25 | 1036.20±34.40 |
| 10 | 926.52±32.66 | 948.50±17.59 |
| 25 | 301.59±4.78 | 888.10±11.75 |
| 50 | 201.57±2.67 | 963.60±14.94 |
| 75 | 191.73±1.91 | 1108.20±11.40 |
| 100 | 195.51±1.88 | 1253.30±12.04 |

| Data for the 100x100 deterministic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 48149.62±0.00† | 12249.00±0.00 |
| 1 | 24723.38±9384.27 | 29241.00±40555.01 |
| 5 | 3333.64±157.31 | 1224.60±48.25 |
| 10 | 1414.77±40.05 | 1153.40±23.54 |
| 25 | 587.50±14.72 | 1227.60±23.58 |
| 50 | 380.49±5.22 | 1366.80±19.37 |
| 75 | 354.12±6.59 | 1602.70±29.52 |
| 100 | 361.38±3.37 | 1862.80±17.39 |

† Based on only 1 run.

| Data for the 150x150 deterministic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | 15427.76±535.52 | 4423.40±900.51 |
| 10 | 7329.74±238.27 | 3524.30±106.59 |
| 25 | 2935.33±60.26 | 3564.60±87.61 |
| 50 | 2080.27±37.05 | 4611.20±99.56 |
| 75 | 2028.04±25.75 | 5930.40±78.65 |
| 100 | 2543.01±30.98 | 8649.70±105.37 |

| Data for the 200x200 deterministic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | 35495.56±1972.87 | 7679.00±3855.68 |
| 10 | 15858.48±383.80 | 5165.20±136.65 |
| 25 | 5640.45±78.36 | 4935.70±63.65 |
| 50 | 3225.79±17.25 | 5285.20±26.28 |
| 75 | 2867.48±32.66 | 6231.20±73.66 |
| 100 | 3372.56±22.79 | 8559.80±57.85 |

| Data for the 20x20 stochastic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | 12872.83±13031.29 | 218758.50±221358.56 |
| 25 | 107.15±42.30 | 1766.30±755.83 |
| 50 | 12.47±0.65 | 180.50±9.71 |
| 75 | 8.95±0.36 | 156.30±6.14 |
| 100 | 7.72±0.33 | 149.60±6.75 |

| Data for the 40x40 stochastic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | 11820.67±7498.85 | 103624.20±66590.87 |
| 25 | 332.79±90.83 | 2508.30±763.49 |
| 50 | 72.13±2.95 | 526.40±24.85 |
| 75 | 63.01±1.18 | 543.30±10.75 |
| 100 | 56.35±1.79 | 535.90±16.40 |

| Data for the 60x60 stochastic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | did not complete | |
| 25 | 874.82±267.89 | 4278.67±1548.59 |
| 50 | 226.12±5.98 | 1081.80±34.10 |
| 75 | 185.43±2.40 | 1052.20±13.14 |
| 100 | 155.25±2.46 | 974.80±15.03 |

| Data for the 80x80 stochastic maze with the optimal trainer | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | did not complete | |
| 25 | 874.57±131.16 | 2579.90±552.58 |
| 50 | 367.46±3.22 | 1267.60±10.84 |
| 75 | 310.72±2.87 | 1263.00±11.72 |
| 100 | 280.71±1.56 | 1254.90±5.99 |

The following tables give the data for the maze experiment with the suboptimal trainers, which is discussed in Section 5.3 starting on page 89. In all of the tables, the column labeled "Actions" shows the mean and 95% confidence interval for the number of actions needed to achieve the stopping criterion, in 1,000's. "Trials" shows the mean and confidence interval for the number of trials performed.

| Data for the 20x20 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 135.81±38.34 | 608.78±730.64 |
| 1 | 145.69±93.19 | 1551.20±2147.05 |
| 5 | 47.77±8.12 | 268.10±175.25 |
| 10 | 27.87±1.79 | 159.60±13.69 |
| 25 | 13.92±0.66 | 155.80±9.87 |
| 50 | 8.67±0.39 | 149.40±6.25 |
| 75 | 7.75±0.21 | 164.00±4.32 |
| 100 | 7.88±0.21 | 184.50±4.44 |

| Data for the 20x20 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 135.81±38.34 | 608.78±730.64 |
| 1 | 127.61±61.86 | 944.20±1574.46 |
| 5 | 56.49±4.58 | 146.10±18.59 |
| 10 | 33.44±2.07 | 151.60±9.57 |
| 25 | 18.16±1.14 | 156.00±9.00 |
| 50 | 11.11±0.38 | 147.70±5.60 |
| 75 | 9.62±0.27 | 155.60±4.61 |
| 100 | 10.03±0.44 | 170.00±6.94 |

| Data for the 20x20 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 135.81±38.34 | 608.78±730.64 |
| 1 | 97.17±8.09 | 146.60±13.80 |
| 5 | 120.02±58.29 | 979.20±1308.87 |
| 10 | 55.59±2.91 | 138.60±12.17 |
| 25 | 33.42±1.50 | 141.60±6.41 |
| 50 | 22.21±1.59 | 151.60±12.54 |
| 75 | 19.16±0.79 | 140.10±6.10 |
| 100 | 21.40±1.00 | 129.10±5.68 |

| Data for the 20x20 deterministic maze with the 60% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 135.81±38.34 | 608.78±730.64 |
| 1 | 110.67±14.93 | 127.40±9.19 |
| 5 | 108.79±12.01 | 213.10±155.21 |
| 10 | 84.24±11.29 | 153.90±9.20 |
| 25 | 50.69±6.30 | 132.30±11.42 |
| 50 | 39.50±3.47 | 147.90±12.19 |
| 75 | 33.72±2.13 | 126.30±6.33 |
| 100 | 44.62±4.43 | 90.20±7.53 |

| Data for the 20x20 deterministic maze with the 75% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 135.81±38.34 | 608.78±730.64 |
| 1 | 141.90±26.30 | 567.40±649.73 |
| 5 | 162.19±76.72 | 902.90±1568.02 |
| 10 | 155.87±20.22 | 143.10±10.64 |
| 25 | 310.85±48.27 | 133.30±12.99 |
| 50 | 1679.33±195.43 | 110.80±7.40 |
| 75 | did not complete | |
| 100 | did not complete | |

| Data for the 40x40 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 1800.69±561.96 | 3667.56±7206.77 |
| 1 | 1512.04±514.92 | 6461.33±5953.52 |
| 5 | 331.32±18.81 | 320.50±43.23 |
| 10 | 154.55±7.66 | 295.00±16.53 |
| 25 | 57.81±2.14 | 288.80±9.08 |
| 50 | 37.69±1.39 | 325.30±13.84 |
| 75 | 33.82±1.27 | 349.90±14.77 |
| 100 | 33.81±1.49 | 380.80±17.13 |

| Data for the 40x40 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 1800.69±561.96 | 3667.56±7206.77 |
| 1 | 1172.30±97.11 | 442.30±265.05 |
| 5 | 434.61±26.74 | 327.90±21.83 |
| 10 | 217.26±13.40 | 302.30±18.75 |
| 25 | 84.58±3.22 | 303.10±17.57 |
| 50 | 47.39±1.80 | 307.60±11.95 |
| 75 | 42.27±1.12 | 323.20±10.87 |
| 100 | 46.38±2.14 | 364.60±16.06 |

| Data for the 40x40 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 1800.69±561.96 | 3667.56±7206.77 |
| 1 | 1511.85±234.97 | 1551.70±2756.50 |
| 5 | 1326.07±760.71 | 5614.37±8261.12 |
| 10 | 593.18±87.99 | 753.30±961.30 |
| 25 | 210.40±12.99 | 304.00±22.10 |
| 50 | 114.79±3.99 | 289.50±11.75 |
| 75 | 96.58±4.52 | 287.20±19.12 |
| 100 | 129.20±5.34 | 300.80±13.66 |

| Data for the 40x40 deterministic maze with the 60% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 1800.69±561.96 | 3667.56±7206.77 |
| 1 | 3011.30±1414.62 | 16816.20±16148.92 |
| 5 | 1291.79±223.67 | 2200.30±2336.19 |
| 10 | 859.14±80.26 | 324.90±26.27 |
| 25 | 673.79±286.91 | 1961.30±2402.26 |
| 50 | 330.56±52.29 | 387.60±210.15 |
| 75 | 244.19±17.84 | 285.20±29.74 |
| 100 | 479.53±45.39 | 220.60±19.11 |

| Data for the 40x40 deterministic maze with the 75% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 1800.69±561.96 | 3667.56±7206.77 |
| 1 | 1972.52±508.87 | 3769.20±5724.91 |
| 5 | 3404.76±1012.96 | 12216.40±11176.83 |
| 10 | 6362.92±5380.71 | 28335.67±55915.37 |
| 25 | did not complete | |
| 50 | did not complete | |
| 75 | did not complete | |
| 100 | did not complete | |

| Data for the 60x60 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 5963.28±1525.81 | 7104.30±9407.78 |
| 1 | 7928.06±10581.04 | 35716.40±77382.77 |
| 5 | 1160.44±160.88 | 1323.40±1151.37 |
| 10 | 590.42±36.15 | 726.60±25.13 |
| 25 | 222.37±5.95 | 717.90±25.45 |
| 50 | 136.19±2.34 | 740.60±11.49 |
| 75 | 142.75±2.80 | 948.70±20.43 |
| 100 | 183.89±3.47 | 1333.50±25.06 |

| Data for the 60x60 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 5963.28±1525.81 | 7104.30±9407.78 |
| 1 | 6224.56±3307.07 | 20476.30±24369.46 |
| 5 | 1502.06±100.84 | 732.20±45.33 |
| 10 | 807.09±51.02 | 729.80±39.15 |
| 25 | 299.22±12.78 | 697.70±17.87 |
| 50 | 180.87±5.33 | 730.30±21.09 |
| 75 | 181.41±2.49 | 880.30±11.97 |
| 100 | 255.25±4.25 | 1277.80±19.60 |

| Data for the 60x60 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 5963.28±1525.81 | 7104.30±9407.78 |
| 1 | 6450.54±3789.75 | 15081.40±27286.38 |
| 5 | 2788.53±120.36 | 983.70±284.00 |
| 10 | 1922.85±187.61 | 1203.20±721.34 |
| 25 | 848.09±45.54 | 792.30±196.63 |
| 50 | 438.48±12.30 | 676.00±17.46 |
| 75 | 375.41±10.30 | 689.20±19.76 |
| 100 | 657.03±16.93 | 925.10±24.83 |

| Data for the 60x60 deterministic maze with the 60% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 5963.28±1525.81 | 7104.30±9407.78 |
| 1 | 11225.42±9933.66 | 46025.00±73592.89 |
| 5 | 7370.22±6628.56 | 22410.20±46698.63 |
| 10 | 5666.05±4227.22 | 16315.40±27827.41 |
| 25 | 3204.68±1726.07 | 6013.30±8927.49 |
| 50 | 1376.67±70.38 | 688.70±26.25 |
| 75 | 1111.22±32.08 | 573.80±15.97 |
| 100 | 2260.73±86.02 | 503.40±16.09 |

| Data for the 80x80 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 29745.13±9581.29 | 47278.86±46996.19 |
| 1 | 13650.27±3970.44 | 14459.30±21180.61 |
| 5 | 2767.40±77.67 | 1036.10±18.99 |
| 10 | 1183.13±37.89 | 976.20±19.39 |
| 25 | 353.59±7.34 | 871.90±8.89 |
| 50 | 227.57±1.44 | 945.70±8.84 |
| 75 | 215.12±2.04 | 1063.70±10.84 |
| 100 | 226.24±2.70 | 1208.40±13.17 |

| Data for the 80x80 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 29745.13±9581.29 | 47278.86±46996.19 |
| 1 | 14507.44±4274.72 | 10189.00±20573.18 |
| 5 | 3761.80±112.68 | 1040.90±32.15 |
| 10 | 1730.43±101.99 | 1002.70±33.97 |
| 25 | 500.15±15.73 | 886.70±11.14 |
| 50 | 286.30±4.68 | 902.60±15.29 |
| 75 | 268.82±3.39 | 991.50±15.15 |
| 100 | 306.26±3.81 | 1148.80±12.95 |

| Data for the 80x80 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 29745.13±9581.29 | 47278.86±46996.19 |
| 1 | 19082.41±3267.56 | 9234.12±17009.38 |
| 5 | 8767.85±826.19 | 1688.78±1042.17 |
| 10 | 4463.65±191.44 | 1040.00±28.27 |
| 25 | 1591.66±54.97 | 962.10±20.97 |
| 50 | 676.25±17.70 | 850.10±15.84 |
| 75 | 552.10±12.41 | 842.40±16.29 |
| 100 | 775.66±11.77 | 919.20±13.47 |

| Data for the 80x80 deterministic maze with the 60% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 29745.13±9581.29 | 47278.86±46996.19 |
| 1 | 20488.96±2210.28 | 1234.25±547.16 |
| 5 | 14726.46±1362.97 | 5139.62±6270.45 |
| 10 | 9513.71±502.22 | 1138.20±90.69 |
| 25 | 4313.83±258.70 | 1024.80±38.95 |
| 50 | 1940.31±78.75 | 895.50±23.54 |
| 75 | 1397.10±67.36 | 785.20±24.58 |
| 100 | 2327.35±59.63 | 648.80±16.21 |

| Data for the 100x100 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 48149.62±0.00 | 12249.00±0.00 |
| 1 | 20189.94±3756.88 | 1800.33±1787.08 |
| 5 | 4087.40±291.14 | 1760.70±769.36 |
| 10 | 1707.41±55.41 | 1179.50±27.96 |
| 25 | 673.28±18.50 | 1205.40±26.94 |
| 50 | 435.33±6.03 | 1350.40±20.92 |
| 75 | 403.91±5.90 | 1560.00±22.40 |
| 100 | 418.03±4.82 | 1800.80±20.17 |

| Data for the 100x100 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 48149.62±0.00 | 12249.00±0.00 |
| 1 | 26004.84±1764.05 | 2099.75±737.11 |
| 5 | 6695.81±1222.88 | 4942.20±4865.28 |
| 10 | 2551.22±192.02 | 1617.10±923.52 |
| 25 | 886.27±22.05 | 1171.30±24.01 |
| 50 | 547.49±10.37 | 1276.00±31.36 |
| 75 | 512.36±5.56 | 1469.30±16.32 |
| 100 | 561.52±8.19 | 1708.70±26.68 |

| Data for the 100x100 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 48149.62±0.00 | 12249.00±0.00 |
| 1 | 39029.33±8414.32 | 6776.25±10177.94 |
| 5 | 16146.92±2007.90 | 6230.44±6357.57 |
| 10 | 9944.00±6165.42 | 12958.40±24811.71 |
| 25 | 2364.61±68.04 | 1190.90±23.82 |
| 50 | 1166.40±32.29 | 1151.20±25.48 |
| 75 | 1008.27±30.77 | 1205.30±40.87 |
| 100 | 1418.29±25.43 | 1429.50±25.68 |

| Data for the 100x100 deterministic maze with the 60% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 48149.62±0.00 | 12249.00±0.00 |
| 1 | 44157.03±52217.52 | 1401.50±2484.02 |
| 5 | 30194.11±5663.58 | 11709.43±20711.99 |
| 10 | 18354.77±4616.79 | 7226.20±14678.62 |
| 25 | 7890.90±2067.81 | 5939.33±6402.69 |
| 50 | 3241.33±133.06 | 1319.20±213.68 |
| 75 | 2388.50±39.10 | 1184.80±25.90 |
| 100 | 4235.08±122.08 | 1170.00±33.40 |

| Data for the 150x150 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | 19033.84±1556.04 | 5109.90±3464.25 |
| 10 | 8939.48±312.09 | 3562.70±133.60 |
| 25 | 3458.85±84.33 | 3521.20±84.70 |
| 50 | 2330.96±59.83 | 4360.80±118.30 |
| 75 | 2296.07±44.01 | 5655.10±117.14 |
| 100 | 2999.11±48.83 | 8450.50±138.59 |

| Data for the 150x150 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | 25797.91±1099.58 | 4052.30±869.29 |
| 10 | 12949.05±430.01 | 3741.40±180.41 |
| 25 | 4710.78±135.42 | 3492.40±132.44 |
| 50 | 2991.54±27.99 | 4097.80±55.79 |
| 75 | 2912.98±44.86 | 5170.90±87.12 |
| 100 | 4202.90±43.35 | 8195.10±83.87 |

| Data for the 150x150 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | 35674.99±1467.21 | 5501.30±2972.19 |
| 25 | 14506.60±503.64 | 3781.70±286.99 |
| 50 | 7409.63±143.39 | 3574.50±92.58 |
| 75 | 6301.79±130.05 | 3912.40±93.94 |
| 100 | 12487.38±120.60 | 6836.60±64.29 |

| Data for the 150x150 deterministic maze with the 60% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | did not complete | |
| 25 | 48260.11±1468.44 | 3717.71±801.09 |
| 50 | 31065.47±2485.20 | 5193.80±3465.62 |
| 75 | 22945.59±487.62 | 3383.30±154.34 |
| 100 | 39774.92±764.25 | 2844.50±57.41 |

| Data for the 200x200 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | 40573.33±1596.30 | 5518.00±452.31 |
| 10 | 19329.77±673.02 | 5227.70±202.69 |
| 25 | 6829.20±119.30 | 4994.30±80.70 |
| 50 | 3706.94±51.49 | 5141.70±78.03 |
| 75 | 3334.47±56.42 | 6091.40±107.89 |
| 100 | 4126.08±31.89 | 8685.80±66.24 |

| Data for the 200x200 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | 28172.10±1030.40 | 5584.90±349.11 |
| 25 | 9648.49±265.87 | 4994.00±168.82 |
| 50 | 4983.84±60.94 | 5012.00±74.50 |
| 75 | 4451.67±36.15 | 5870.30±56.61 |
| 100 | 6188.33±61.42 | 9044.20±87.95 |

| Data for the 200x200 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | did not complete | |
| 25 | 30668.63±1000.80 | 5367.56±256.65 |
| 50 | 13979.77±532.59 | 4848.90±202.31 |
| 75 | 10473.81±215.33 | 4907.10±105.08 |
| 100 | 19938.72±251.05 | 8426.80±106.50 |

| Data for the 200x200 deterministic maze with the 60% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | did not complete | |
| 25 | did not complete | |
| 50 | did not complete | |
| 75 | 36921.41±1965.57 | 4090.10±208.18 |
| 100 | did not complete | |

| Data for the 20x20 stochastic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | 8492.60±9317.23 | 142470.00±155592.98 |
| 25 | 142.61±79.89 | 2265.70±1366.98 |
| 50 | 20.85±14.30 | 297.40±242.39 |
| 75 | 10.86±0.45 | 164.50±6.79 |
| 100 | 8.95±0.34 | 150.40±5.65 |

| Data for the 20x20 stochastic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | 849.64±0.00 | 14026.00±0.00 |
| 25 | 344.88±233.25 | 5263.50±3688.45 |
| 50 | 89.91±49.80 | 1269.90±745.93 |
| 75 | 14.18±0.98 | 172.30±11.58 |
| 100 | 12.40±0.54 | 157.40±6.43 |

| Data for the 40x40 stochastic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | 31074.17±41695.51 | 270149.67±364706.94 |
| 25 | 3440.30±2081.90 | 28964.60±17822.12 |
| 50 | 82.54±2.16 | 535.90±17.36 |
| 75 | 72.70±1.71 | 548.50±13.01 |
| 100 | 66.09±2.65 | 536.30±21.77 |

| Data for the 40x40 stochastic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | 16911.45±10161.31 | 141762.00±87252.10 |
| 25 | 21228.99±16430.12 | 170301.62±132418.49 |
| 50 | 100.98±5.46 | 534.40±37.64 |
| 75 | 91.94±2.89 | 547.70±18.53 |
| 100 | 95.89±2.98 | 579.40±18.14 |

| Data for the 60x60 stochastic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | did not complete | |
| 25 | did not complete | |
| 50 | 261.78±12.67 | 1129.40±66.24 |
| 75 | 219.67±3.51 | 1097.10±17.76 |
| 100 | 188.30±2.51 | 1015.00±12.84 |

| Data for the 60x60 stochastic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | did not complete | |
| 25 | did not complete | |
| 50 | 354.81±35.75 | 1302.67±153.59 |
| 75 | 308.33±17.04 | 1223.60±74.99 |
| 100 | 269.45±2.97 | 1091.10±12.02 |

| Data for the 80x80 stochastic maze with the 10% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | did not complete | |
| 25 | 925.39±509.51 | 2426.67±1989.61 |
| 50 | 408.81±3.65 | 1253.20±13.55 |
| 75 | 356.89±3.89 | 1269.30±15.15 |
| 100 | 334.82±3.08 | 1275.30±12.20 |

| Data for the 80x80 stochastic maze with the 25% error rate | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | did not complete | |
| 1 | did not complete | |
| 5 | did not complete | |
| 10 | did not complete | |
| 25 | did not complete | |
| 50 | 516.27±12.20 | 1284.60±40.23 |
| 75 | 466.86±14.48 | 1305.87±44.55 |
| 100 | 512.50±31.37 | 1449.37±87.79 |

The following tables give the data for the race track task. The information provided here represents the results of the experiment described in Section 5.4, which starts on page 97. The first column shows the Asking Rate, as a percentage. The column labeled "Actions" shows the mean and 95% confidence interval for the number of actions necessary to achieve the stopping criterion.

| Data for the First Track | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 65.21±8.95 | 4861.60±594.54 |
| 1 | 63.09±11.33 | 4656.15±717.59 |
| 5 | 59.82±11.30 | 4421.20±746.12 |
| 10 | 54.72±9.61 | 4078.90±645.03 |
| 20 | 38.50±8.39 | 2912.20±614.66 |
| 30 | 25.48±7.04 | 1894.40±503.85 |
| 40 | 15.71±2.74 | 1126.25±195.48 |
| 50 | 13.47±2.06 | 902.75±140.38 |
| 60 | 10.29±1.39 | 625.00±84.19 |
| 70 | 6.76±1.39 | 362.75±75.87 |
| 80 | 8.15±2.43 | 376.95±111.98 |
| 85 | 27.65±10.87 | 1176.55±460.24 |
| 90 | 108.82±49.23 | 4448.05±961.05 |
| 95 | 428.47±0.00 | 15255.57±0.00 |

| Data for the Second Track | | |
|---|---|---|
| Asking Rate | Actions | Trials |
| 0% | 338.35±83.61 | 15495.35±3108.14 |
| 1 | 277.59±51.81 | 13232.70±1920.43 |
| 5 | 236.12±50.99 | 11759.55±1955.59 |
| 10 | 177.56±29.62 | 9585.90±1167.43 |
| 20 | 117.86±11.45 | 7206.65±483.29 |
| 30 | 106.42±11.87 | 6614.45±551.24 |
| 40 | 78.06±13.82 | 4976.65±761.76 |
| 50 | 54.27±8.42 | 3439.75±496.13 |
| 60 | 25.50±3.90 | 1491.95±229.29 |
| 70 | 16.57±2.65 | 821.90±131.32 |
| 80 | 17.17±3.52 | 655.65±133.41 |
| 85 | 44.49±12.58 | 1433.80±398.59 |
| 90 | 123.69±31.49 | 3320.11±843.55 |
| 95 | did not complete | |

The following tables give the data for the maze experiment with the uncertainty asking strategy, which is discussed in Section 5.5 starting on page 102. In all of the tables, the column labeled "Actions" shows the mean and 95% confidence interval for the number of actions needed to achieve the stopping criterion, in 1,000's. "Trials" shows the mean and confidence interval for the number of trials performed.

| Data for the 20x20 deterministic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 135.81±38.34 | 608.78±730.64 |
| 0.0 | 9.47±0.57 | 179.60±8.42 |
| 0.0001 | 5.86±0.21 | 158.20±5.69 |
| 0.001 | 6.13±0.21 | 165.80±5.71 |
| 0.01 | 6.46±0.10 | 175.30±2.70 |
| 0.1 | 6.80±0.26 | 185.40±7.20 |
| 0.25 | 6.72±0.17 | 183.70±4.54 |
| 0.50 | 6.66±0.22 | 182.70±5.90 |
| 0.75 | 6.65±0.28 | 182.80±7.64 |
| 1.0 | 6.61±0.16 | 183.50±4.45 |

| Data for the 20x20 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 135.81±38.34 | 608.78±730.64 |
| 0.0 | 16.53±16.62 | 338.20±389.66 |
| 0.0001 | 6.51±0.18 | 160.70±4.25 |
| 0.001 | 6.92±0.18 | 169.90±4.69 |
| 0.01 | 7.14±0.19 | 173.90±4.69 |
| 0.1 | 7.32±0.28 | 176.60±6.76 |
| 0.25 | 7.53±0.38 | 180.70±9.20 |
| 0.50 | 7.62±0.28 | 182.50±6.93 |
| 0.75 | 7.21±0.33 | 171.50±7.70 |
| 1.0 | 7.67±0.15 | 179.00±3.60 |

| Data for the 20x20 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 135.81±38.34 | 608.78±730.64 |
| 0.0 | 9.38±0.57 | 150.80±9.25 |
| 0.0001 | 7.85±0.29 | 159.20±5.20 |
| 0.001 | 8.56±0.24 | 169.00±5.67 |
| 0.01 | 8.42±0.28 | 163.30±5.09 |
| 0.1 | 9.05±0.14 | 168.70±2.52 |
| 0.25 | 9.10±0.40 | 168.00±6.73 |
| 0.50 | 9.49±0.21 | 169.50±4.78 |
| 0.75 | 9.73±0.37 | 170.10±6.83 |
| 1.0 | 10.17±0.32 | 169.40±4.12 |

| Data for the 20x20 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 135.81±38.34 | 608.78±730.64 |
| 0.0 | 13.31±0.71 | 139.20±5.72 |
| 0.0001 | 17.82±1.06 | 160.20±6.50 |
| 0.001 | 18.15±0.73 | 152.60±6.70 |
| 0.01 | 19.07±0.84 | 150.40±7.30 |
| 0.1 | 18.73±0.86 | 134.90±8.60 |
| 0.25 | 19.14±1.01 | 133.50±8.47 |
| 0.50 | 20.72±1.41 | 135.70±10.74 |
| 0.75 | 20.97±0.83 | 132.10±5.28 |
| 1.0 | 22.23±1.51 | 135.30±8.56 |

| Data for the 40x40 deterministic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 1800.69±561.96 | 3667.56±7206.77 |
| 0.0 | 30.72±2.23 | 260.50±17.06 |
| 0.0001 | 23.19±0.91 | 305.20±11.83 |
| 0.001 | 24.61±0.94 | 324.40±12.26 |
| 0.01 | 28.42±1.26 | 375.10±16.26 |
| 0.1 | 27.36±1.04 | 363.00±13.76 |
| 0.25 | 29.94±1.34 | 397.80±17.53 |
| 0.50 | 29.08±1.51 | 387.90±19.73 |
| 0.75 | 29.00±1.51 | 388.70±20.18 |
| 1.0 | 30.81±1.87 | 416.30±25.31 |

| Data for the 40x40 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 1800.69±561.96 | 3667.56±7206.77 |
| 0.0 | 31.50±2.33 | 264.10±17.79 |
| 0.0001 | 26.77±1.12 | 317.50±14.05 |
| 0.001 | 29.59±0.99 | 348.40±12.16 |
| 0.01 | 30.67±1.13 | 360.20±13.82 |
| 0.1 | 32.54±1.64 | 378.50±19.60 |
| 0.25 | 32.71±0.54 | 378.50±7.11 |
| 0.50 | 33.24±1.37 | 382.10±16.29 |
| 0.75 | 34.47±1.73 | 391.90±19.99 |
| 1.0 | 34.09±1.98 | 383.30±22.09 |

| Data for the 40x40 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 1800.69±561.96 | 3667.56±7206.77 |
| 0.0 | 35.19±2.20 | 272.80±19.44 |
| 0.0001 | 35.94±0.90 | 335.10±9.20 |
| 0.001 | 36.65±1.03 | 335.80±11.56 |
| 0.01 | 39.52±1.22 | 354.90±11.77 |
| 0.1 | 42.40±1.48 | 369.20±13.98 |
| 0.25 | 42.38±1.61 | 362.40±14.30 |
| 0.50 | 44.03±2.76 | 363.00±23.98 |
| 0.75 | 44.92±1.76 | 361.80±14.09 |
| 1.0 | 47.13±1.98 | 368.40±15.87 |

| Data for the 40x40 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 1800.69±561.96 | 3667.56±7206.77 |
| 0.0 | 150.32±208.68 | 1348.60±2428.76 |
| 0.0001 | 105.69±5.32 | 347.80±21.76 |
| 0.001 | 110.09±5.29 | 343.70±19.30 |
| 0.01 | 111.12±3.97 | 326.80±14.48 |
| 0.1 | 119.71±6.52 | 326.30±17.71 |
| 0.25 | 119.89±4.43 | 315.80±15.04 |
| 0.50 | 126.71±5.96 | 313.90±14.80 |
| 0.75 | 126.55±5.05 | 302.40±12.24 |
| 1.0 | 133.60±5.76 | 310.90±13.43 |

| Data for the 60x60 deterministic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 5963.28±1525.81 | 7104.30±9407.78 |
| 0.0 | 149.49±25.92 | 820.60±182.67 |
| 0.0001 | 103.33±2.48 | 879.90±21.17 |
| 0.001 | 112.44±1.71 | 957.80±14.69 |
| 0.01 | 129.20±2.83 | 1101.40±23.87 |
| 0.1 | 151.36±3.29 | 1293.70±28.20 |
| 0.25 | 155.31±1.99 | 1332.70±17.17 |
| 0.50 | 154.81±1.92 | 1337.80±16.66 |
| 0.75 | 153.63±2.25 | 1340.70±19.58 |
| 1.0 | 152.45±1.84 | 1337.30±16.12 |

| Data for the 60x60 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 5963.28±1525.81 | 7104.30±9407.78 |
| 0.0 | 252.18±256.94 | 1625.50±1921.70 |
| 0.0001 | 114.61±2.22 | 883.10±16.58 |
| 0.001 | 126.81±2.18 | 975.60±16.75 |
| 0.01 | 143.28±1.41 | 1100.80±10.61 |
| 0.1 | 170.98±3.01 | 1304.50±23.32 |
| 0.25 | 172.98±3.38 | 1308.30±25.48 |
| 0.50 | 178.77±1.78 | 1330.60±13.17 |
| 0.75 | 180.03±2.82 | 1313.70±20.09 |
| 1.0 | 181.04±2.00 | 1312.40±14.67 |

| Data for the 60x60 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 5963.28±1525.81 | 7104.30±9407.78 |
| 0.0 | 243.81±272.08 | 1542.50±2078.83 |
| 0.0001 | 147.47±2.25 | 910.50±15.04 |
| 0.001 | 163.77±2.65 | 1004.40±16.23 |
| 0.01 | 184.25±2.75 | 1120.00±17.60 |
| 0.1 | 213.41±3.98 | 1259.50±23.85 |
| 0.25 | 219.60±4.97 | 1252.90±31.34 |
| 0.50 | 239.67±3.94 | 1285.70±20.69 |
| 0.75 | 252.22±3.77 | 1278.20±21.01 |
| 1.0 | 252.41±3.52 | 1265.10±17.14 |

| Data for the 60x60 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 5963.28±1525.81 | 7104.30±9407.78 |
| 0.0 | 236.35±8.25 | 657.30±33.35 |
| 0.0001 | 468.16±11.01 | 1042.20±19.75 |
| 0.001 | 509.37±15.99 | 1057.70±34.50 |
| 0.01 | 533.27±11.97 | 1024.70±30.71 |
| 0.1 | 546.00±7.42 | 981.30±20.16 |
| 0.25 | 550.86±13.97 | 950.20±23.14 |
| 0.50 | 625.07±18.27 | 941.50±25.28 |
| 0.75 | 654.48±13.17 | 931.40±18.74 |
| 1.0 | 653.04±13.55 | 917.70±17.23 |

| Data for the 80x80 deterministic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 29745.13±9581.29 | 47278.86±46996.19 |
| 0.0 | 182.39±3.15 | 767.40±9.62 |
| 0.0001 | 150.46±1.44 | 934.50±8.68 |
| 0.001 | 161.15±1.40 | 1001.10±8.54 |
| 0.01 | 173.56±2.59 | 1079.40±16.07 |
| 0.1 | 190.54±1.76 | 1188.60±10.86 |
| 0.25 | 195.62±2.52 | 1226.90±15.82 |
| 0.50 | 195.59±1.39 | 1239.70±8.76 |
| 0.75 | 196.13±3.27 | 1252.90±20.98 |
| 1.0 | 194.81±1.35 | 1248.80±8.64 |

| Data for the 80x80 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 29745.13±9581.29 | 47278.86±46996.19 |
| 0.0 | 185.91±2.60 | 766.10±7.31 |
| 0.0001 | 165.81±1.98 | 942.50±11.08 |
| 0.001 | 175.45±1.82 | 994.80±10.13 |
| 0.01 | 190.17±1.28 | 1076.60±7.27 |
| 0.1 | 208.23±1.70 | 1171.20±10.36 |
| 0.25 | 215.81±4.24 | 1200.70±23.62 |
| 0.50 | 225.12±2.97 | 1221.90±15.45 |
| 0.75 | 227.71±2.59 | 1222.40±13.79 |
| 1.0 | 228.41±2.28 | 1219.30±12.04 |

| Data for the 80x80 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 29745.13±9581.29 | 47278.86±46996.19 |
| 0.0 | 199.80±3.81 | 768.60±12.47 |
| 0.0001 | 204.30±2.60 | 953.40±12.87 |
| 0.001 | 220.80±2.56 | 1024.80±12.14 |
| 0.01 | 234.72±2.12 | 1076.60±11.82 |
| 0.1 | 256.88±3.61 | 1145.70±17.41 |
| 0.25 | 272.15±2.32 | 1155.40±9.16 |
| 0.50 | 296.19±3.96 | 1157.20±13.34 |
| 0.75 | 304.31±4.17 | 1154.60±15.41 |
| 1.0 | 305.52±3.16 | 1145.60±12.80 |

| Data for the 80x80 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 29745.13±9581.29 | 47278.86±46996.19 |
| 0.0 | 318.49±7.17 | 784.20±12.26 |
| 0.0001 | 537.42±8.08 | 1029.30±16.24 |
| 0.001 | 568.81±4.43 | 1025.30±12.17 |
| 0.01 | 596.69±8.51 | 996.90±11.13 |
| 0.1 | 619.15±9.90 | 958.80±15.24 |
| 0.25 | 685.62±6.32 | 937.00±9.09 |
| 0.50 | 740.37±13.37 | 937.80±15.64 |
| 0.75 | 788.68±14.43 | 939.10±16.59 |
| 1.0 | 785.61±11.57 | 927.50±13.41 |

| Data for the 100x100 deterministic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 48149.62±0.00† | 12249.00±0.00 |
| 0.0 | 436.10±13.24 | 1365.30±37.13 |
| 0.0001 | 256.96±3.53 | 1290.90±17.89 |
| 0.001 | 275.42±3.04 | 1384.40±15.60 |
| 0.01 | 301.15±2.11 | 1515.10±10.71 |
| 0.1 | 344.28±4.84 | 1738.70±24.51 |
| 0.25 | 355.78±4.26 | 1806.30±21.74 |
| 0.50 | 363.70±2.43 | 1859.50±12.18 |
| 0.75 | 359.01±2.94 | 1845.10±15.13 |
| 1.0 | 363.13±3.87 | 1871.80±19.95 |

† Based on only 1 run.

| Data for the 100x100 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 48149.62±0.00† | 12249.00±0.00 |
| 0.0 | 405.49±11.47 | 1253.10±32.98 |
| 0.0001 | 284.70±3.87 | 1294.00±17.31 |
| 0.001 | 304.67±3.85 | 1381.60±16.79 |
| 0.01 | 337.60±3.89 | 1526.90±17.94 |
| 0.1 | 377.43±6.60 | 1693.90±29.06 |
| 0.25 | 397.87±5.64 | 1763.60±25.41 |
| 0.50 | 414.46±4.37 | 1810.30±18.20 |
| 0.75 | 413.49±3.92 | 1788.00±16.89 |
| 1.0 | 420.55±5.12 | 1810.90±22.25 |

† Based on only 1 run.

| Data for the 100x100 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 48149.62±0.00† | 12249.00±0.00 |
| 0.0 | 396.12±10.41 | 1150.80±29.55 |
| 0.0001 | 356.27±4.94 | 1303.90±18.43 |
| 0.001 | 393.44±4.29 | 1427.90±13.86 |
| 0.01 | 424.52±9.79 | 1521.60±33.91 |
| 0.1 | 478.23±8.11 | 1662.10±27.43 |
| 0.25 | 517.39±8.14 | 1714.10±25.02 |
| 0.50 | 543.69±6.22 | 1709.20±19.41 |
| 0.75 | 560.47±8.24 | 1717.50±25.44 |
| 1.0 | 564.97±7.46 | 1721.50±22.99 |

† Based on only 1 run.

| Data for the 100x100 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | 48149.62±0.00† | 12249.00±0.00 |
| 0.0 | 603.70±32.54 | 1161.50±106.48 |
| 0.0001 | 1010.04±16.05 | 1524.40±19.18 |
| 0.001 | 1069.49±9.04 | 1544.90±13.59 |
| 0.01 | 1128.70±16.23 | 1540.30±25.20 |
| 0.1 | 1191.48±19.81 | 1478.80±25.25 |
| 0.25 | 1298.91±23.36 | 1488.00±26.48 |
| 0.50 | 1395.86±17.32 | 1457.60±20.38 |
| 0.75 | 1427.42±25.09 | 1443.70±26.94 |
| 1.0 | 1423.10±22.01 | 1431.00±21.70 |

† Based on only 1 run.

| Data for the 150x150 deterministic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0 | 1142.20±32.19 | 3080.70±64.23 |
| 0.0001 | 1400.86±14.46 | 4649.40±47.32 |
| 0.001 | 1547.51±37.83 | 5137.40±124.25 |
| 0.01 | 1856.40±29.30 | 6166.00±96.81 |
| 0.1 | 2313.91±25.93 | 7766.00±87.25 |
| 0.25 | 2479.89±29.88 | 8366.20±101.11 |
| 0.50 | 2536.74±27.04 | 8600.30±91.89 |
| 0.75 | 2556.54±20.46 | 8689.30±69.51 |
| 1.0 | 2550.77±22.91 | 8676.10±77.94 |

| Data for the 150x150 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0 | 1171.43±40.16 | 2931.80±89.45 |
| 0.0001 | 1589.75±25.09 | 4720.00±74.35 |
| 0.001 | 1780.46±18.01 | 5281.60±54.13 |
| 0.01 | 2141.62±28.50 | 6347.80±85.60 |
| 0.1 | 2680.86±28.37 | 7763.90±80.38 |
| 0.25 | 2908.75±29.40 | 8334.80±82.72 |
| 0.50 | 2991.54±51.43 | 8478.40±145.65 |
| 0.75 | 3000.70±52.07 | 8463.40±146.63 |
| 1.0 | 3036.09±41.86 | 8554.50±117.25 |

| Data for the 150x150 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0 | 1255.82±28.21 | 2697.20±50.61 |
| 0.0001 | 2130.41±28.94 | 4973.80±69.38 |
| 0.001 | 2396.96±28.63 | 5570.80±72.24 |
| 0.01 | 2827.10±42.42 | 6544.30±104.86 |
| 0.1 | 3702.85±43.07 | 7884.20±88.00 |
| 0.25 | 3978.96±42.51 | 8147.50±85.41 |
| 0.50 | 4144.22±43.79 | 8195.80±85.77 |
| 0.75 | 4213.37±33.44 | 8224.60±63.37 |
| 1.0 | 4251.70±30.78 | 8286.90±58.69 |

| Data for the 150x150 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0 | 3204.69±111.11 | 2942.30±98.29 |
| 0.0001 | 7593.36±114.88 | 6527.90±86.41 |
| 0.001 | 8585.01±116.72 | 7218.90±122.98 |
| 0.01 | 9480.76±125.60 | 7445.90±103.22 |
| 0.1 | 10989.60±233.30 | 7140.60±127.07 |
| 0.25 | 11835.37±209.02 | 6994.10±106.86 |
| 0.50 | 12542.31±157.49 | 6912.60±91.21 |
| 0.75 | 12435.71±197.32 | 6817.70±108.17 |
| 1.0 | 12372.58±214.14 | 6777.80±113.98 |

| Data for the 200x200 deterministic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0 | 2179.25±92.51 | 4179.90±110.58 |
| 0.0001 | 1993.73±25.53 | 4919.90±62.66 |
| 0.001 | 2240.13±26.55 | 5530.10±65.80 |
| 0.01 | 2595.23±19.11 | 6426.30±47.18 |
| 0.1 | 3135.14±26.79 | 7885.40±67.18 |
| 0.25 | 3281.78±33.19 | 8288.40±83.92 |
| 0.50 | 3354.74±35.92 | 8501.90±91.12 |
| 0.75 | 3367.88±35.63 | 8544.00±90.46 |
| 1.0 | 3369.88±41.09 | 8553.00±104.29 |

| Data for the 200x200 deterministic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0 | 1944.05±45.46 | 3579.20±66.27 |
| 0.0001 | 2330.18±29.92 | 5197.80±68.67 |
| 0.001 | 2584.68±33.69 | 5761.20±75.74 |
| 0.01 | 3041.19±54.00 | 6729.80±119.30 |
| 0.1 | 3800.14±31.97 | 8125.00±71.39 |
| 0.25 | 3975.71±29.06 | 8445.50±57.68 |
| 0.50 | 4083.79±33.72 | 8621.70±71.19 |
| 0.75 | 4111.50±64.52 | 8662.40±135.14 |
| 1.0 | 4097.56±39.09 | 8626.70±80.60 |

| Data for the 200x200 deterministic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0 | 2166.07±30.57 | 3457.80±42.41 |
| 0.0001 | 3288.21±26.71 | 5853.70±49.11 |
| 0.001 | 3634.26±66.26 | 6460.70±122.69 |
| 0.01 | 4371.37±85.37 | 7550.80±140.79 |
| 0.1 | 5683.19±62.99 | 8809.10±94.40 |
| 0.25 | 6040.23±71.81 | 9079.40±104.75 |
| 0.50 | 6151.86±36.77 | 9060.60±55.65 |
| 0.75 | 6224.40±57.54 | 9107.90±84.05 |
| 1.0 | 6257.83±51.01 | 9147.00±77.71 |

| Data for the 200x200 deterministic maze with the 50% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0 | 5402.69±202.59 | 3801.50±153.63 |
| 0.0001 | 12390.86±214.79 | 8263.50±169.61 |
| 0.001 | 13458.94±183.89 | 8656.70±103.78 |
| 0.01 | 14952.04±209.95 | 8895.10±135.09 |
| 0.1 | 18507.13±396.92 | 8730.00±181.04 |
| 0.25 | 19556.09±235.42 | 8573.60±99.48 |
| 0.50 | 20035.41±328.14 | 8512.70±136.60 |
| 0.75 | 20134.04±318.32 | 8510.60±137.57 |
| 1.0 | 19853.75±339.89 | 8396.60±141.15 |

| Data for the 20x20 stochastic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 7.70±0.29 | 144.40±5.05 |
| 0.001 | 7.89±0.45 | 149.80±7.83 |
| 0.01 | 8.12±0.35 | 153.20±4.43 |
| 0.1 | 7.58±0.24 | 145.00±4.99 |
| 0.25 | 7.67±0.18 | 147.80±3.63 |
| 0.50 | 7.68±0.34 | 147.90±6.43 |
| 0.75 | 7.91±0.34 | 151.80±6.88 |
| 1.0 | 7.67±0.25 | 149.00±5.26 |

| Data for the 20x20 stochastic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 10.07±2.11 | 174.20±36.79 |
| 0.001 | 8.85±0.33 | 152.80±4.68 |
| 0.01 | 8.68±0.37 | 150.20±6.55 |
| 0.1 | 8.89±0.25 | 152.00±4.96 |
| 0.25 | 8.91±0.30 | 151.10±5.29 |
| 0.50 | 8.89±0.36 | 151.10±5.84 |
| 0.75 | 8.95±0.34 | 151.20±5.70 |
| 1.0 | 8.86±0.38 | 148.60±6.16 |

| Data for the 20x20 stochastic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 26.74±28.07 | 431.20±506.68 |
| 0.001 | 11.65±0.61 | 164.10±5.82 |
| 0.01 | 11.51±0.75 | 163.10±11.17 |
| 0.1 | 11.61±0.51 | 158.80±7.52 |
| 0.25 | 11.68±0.25 | 159.30±2.92 |
| 0.50 | 11.56±0.38 | 153.90±5.06 |
| 0.75 | 12.01±0.48 | 154.30±6.39 |
| 1.0 | 12.13±0.48 | 151.80±6.26 |

| Data for the 40x40 stochastic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 58.92±1.65 | 542.60±14.88 |
| 0.001 | 58.13±0.89 | 536.70±6.74 |
| 0.01 | 57.08±1.60 | 529.10±15.37 |
| 0.1 | 56.43±1.58 | 527.20±15.02 |
| 0.25 | 56.70±0.63 | 530.80±5.89 |
| 0.50 | 56.27±0.89 | 528.30±7.79 |
| 0.75 | 57.23±0.69 | 541.40±7.56 |
| 1.0 | 56.36±1.32 | 534.70±12.14 |

| Data for the 40x40 stochastic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 69.24±2.62 | 584.60±20.45 |
| 0.001 | 67.02±2.15 | 563.40±17.42 |
| 0.01 | 66.25±2.73 | 556.60±23.64 |
| 0.1 | 64.78±2.59 | 541.60±21.56 |
| 0.25 | 65.57±1.33 | 544.60±11.34 |
| 0.50 | 66.73±1.74 | 549.40±13.42 |
| 0.75 | 67.43±1.85 | 550.30±13.73 |
| 1.0 | 65.83±1.81 | 535.20±14.19 |

| Data for the 40x40 stochastic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 90.12±6.80 | 639.90±50.76 |
| 0.001 | 90.11±4.00 | 621.30±25.51 |
| 0.01 | 88.27±2.76 | 600.90±21.42 |
| 0.1 | 88.14±2.08 | 589.10±13.62 |
| 0.25 | 86.19±2.04 | 559.10±13.34 |
| 0.50 | 90.42±2.73 | 568.90±17.39 |
| 0.75 | 97.74±4.36 | 596.10±25.33 |
| 1.0 | 95.66±1.87 | 580.00±11.35 |

| Data for the 60x60 stochastic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 160.29±1.87 | 977.20±11.46 |
| 0.001 | 158.13±3.04 | 966.60±19.03 |
| 0.01 | 158.31±1.89 | 973.20±10.54 |
| 0.1 | 156.57±2.66 | 967.00±16.39 |
| 0.25 | 154.64±1.35 | 958.00±8.48 |
| 0.50 | 153.91±1.51 | 959.10±9.44 |
| 0.75 | 154.17±2.22 | 966.40±13.12 |
| 1.0 | 152.45±2.35 | 957.00±14.07 |

| Data for the 60x60 stochastic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 209.58±66.81 | 1168.80±373.03 |
| 0.001 | 180.59±3.60 | 1009.90±19.89 |
| 0.01 | 181.05±2.43 | 1009.20±13.18 |
| 0.1 | 181.72±2.98 | 1008.90±17.03 |
| 0.25 | 185.29±2.22 | 1017.90±11.57 |
| 0.50 | 185.89±2.33 | 1011.60±12.96 |
| 0.75 | 187.50±2.25 | 1013.10±12.20 |
| 1.0 | 186.88±2.85 | 1005.90±15.40 |

| Data for the 60x60 stochastic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 238.94±14.25 | 1134.00±69.63 |
| 0.001 | 239.67±5.07 | 1117.90±20.11 |
| 0.01 | 242.91±3.88 | 1115.50±16.53 |
| 0.1 | 246.24±3.05 | 1096.60±13.47 |
| 0.25 | 261.62±3.32 | 1119.40±17.26 |
| 0.50 | 267.19±3.45 | 1108.20±13.76 |
| 0.75 | 274.40±4.61 | 1113.50±19.05 |
| 1.0 | 274.61±5.55 | 1110.70±20.99 |

| Data for the 80x80 stochastic maze with the 0% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 287.93±1.78 | 1247.00±8.09 |
| 0.001 | 286.69±2.28 | 1245.10±8.93 |
| 0.01 | 288.39±2.45 | 1257.10±11.32 |
| 0.1 | 284.96±3.53 | 1248.00±15.29 |
| 0.25 | 286.06±3.19 | 1261.00±14.05 |
| 0.50 | 281.13±2.88 | 1251.30±12.85 |
| 0.75 | 280.60±2.59 | 1252.30±10.53 |
| 1.0 | 280.03±3.58 | 1251.80±16.21 |

| Data for the 80x80 stochastic maze with the 10% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 321.82±3.02 | 1286.10±12.89 |
| 0.001 | 322.13±2.43 | 1285.70±10.66 |
| 0.01 | 326.72±3.61 | 1297.80±14.57 |
| 0.1 | 324.17±3.12 | 1278.00±12.37 |
| 0.25 | 329.63±3.25 | 1274.90±11.91 |
| 0.50 | 331.56±3.31 | 1270.00±12.64 |
| 0.75 | 334.67±2.89 | 1279.00±11.35 |
| 1.0 | 337.39±2.87 | 1285.80±10.06 |

| Data for the 80x80 stochastic maze with the 25% error rate | | |
|---|---|---|
| Asking Factor | Actions | Trials |
| Q | did not complete | |
| 0.0001 | 400.95±9.10 | 1364.70±31.24 |
| 0.001 | 396.59±5.74 | 1327.10±18.67 |
| 0.01 | 399.43±4.41 | 1307.80±14.89 |
| 0.1 | 418.22±4.61 | 1300.00±12.78 |
| 0.25 | 445.06±9.51 | 1306.10±28.45 |
| 0.50 | 455.27±6.98 | 1303.90±20.27 |
| 0.75 | 487.31±39.25 | 1380.62±112.22 |
| 1.0 | 499.31±38.35 | 1410.80±106.96 |

# BIBLIOGRAPHY


Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine, 9*, 31-37.

Bareiss, R., & Porter, B. W. (1987). Protos: An exemplar-based learning apprentice. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 12-23). Irvine, CA: Morgan Kaufmann.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics, 13*, 835-846.

Barto, A. G. (1990). Connectionist learning for control: An overview. In Miller, Sutton & Werbos (Eds.), *Neural Networks for Control.* Cambridge, MA: MIT Press.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 72*, 81-138.

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

Berliner, H. J. (1984). Search vs knowledge: An analysis from the domain of games. In Elithorn & Banerji (Eds.), *Artificial and Human Intelligence.* New York: Elsevier Science Publishers.

Clouse, J. A., & Utgoff, P. E. (1992). A teaching method for reinforcement learning. *Machine Learning: Proceedings of the Ninth International Conference* (pp. 92-101). San Mateo, CA: Morgan Kaufmann.

Clouse, J. A. (1995). *Action set approach to reinforcement learning,* (Technical Report 95-108), Amherst, MA: University of Massachusetts, Computer Science Department.

Crites, R. H., & Barto, A. G. (1996). Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems.* Cambridge, MA: MIT Press.

Dayan, P. D. (1992). The convergence of TD($\lambda$) for general $\lambda$. *Machine Learning, 8*, 341-362.

Dayan, P. D., & Sejnowski, T. J. (1994). TD($\lambda$) converges with probability 1. *Machine Learning, 14*, 295-301.

Dietterich, T. G., London, B., Clarkson, K., & Dromey, G. (1982). Learning and inductive inference. In Cohen & Feigenbaum (Eds.), *The Handbook of Artificial Intelligence: Volume III.* San Mateo, CA: Morgan Kaufmann.

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

Gardner, M. (1973). Mathematical games. *Scientific American, 228*, 108.

Ginsberg, M. (1993). *Essentials of artificial intelligence.* Morgan Kaufmann.

Gordon, D., & Subramanian, D. (1994). A multistrategy learning scheme for agent knowledge acquisition. *Informatica, 17*, 331-346.

Gullapalli, V. (1991). A comparison of supervised and reinforcement learning methods on a reinforcement learning task. *Proceedings of the 1991 IEEE International Symposium on Intelligent Control* (pp. 394-399). Arlington, VA.

Gullapalli, V. (1992). *Reinforcement learning and its application to control.* Doctoral dissertation, Computer and Information Science, University of Massachusetts, Amherst, MA.

Hampson, S. E., & Volper, D. J. (1986). Linear function neurons: Structure and training. *Biological Cybernetics, 53*, 203-217.

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

Jaakola, T., Jordan, M. I., & Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation, 6*, ??-??

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237-285.

Korf, R. E. (1988). Search: A survey of recent results. In Howard Shrobe & American Association for Artificial Intelligence (Eds.), *Exploring Artificial Intelligence.* San Mateo: Morgan Kaufman.

Lin, Long-Ji (1991). Programming robots using reinforcement learning and teaching. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 781-786). Anaheim, CA: MIT Press.

Lin, Long-Ji (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning, 8*, 293-321.

Lin, Long-Ji (1993). Scaling up reinforcement learning for robot control. *Machine Learning: Proceedings of the Tenth International Conference* (pp. 182-189). Amherst, MA: Morgan Kaufmann.

Luger, George F., & Stubblefield, William A. (1993). *Artificial intelligence: Structures and strategies for complex problem solving.* Benjamin/Cummings.

Maclin, R., & Shavlik, J. W. (1994). Incorporating advice into agents that learn from reinforcements. *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 694-699). Seattle, WA: MIT Press.

Maclin, R. F. (1995). *Learning from instruction and experiense: Methods for incorporating procedural domain theories into knowledge-based neural networks.* Doctoral dissertation, Computer Science Department, University of Wisconsin, Madison, WI.

Maclin, R., & Shavlik, J. W. (1996). Creating advice-taking reinforcement learners. *Machine Learning, 22*, 251-282.

Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence, 55*, 311-365.

Minsky, M. (1963). Steps towards artificial intelligence. In Feigenbaum & Feldman (Eds.), *Computers and thought.* New York: McGraw-Hill.

Mitchell, T. M., Mahadevan, S., & Steinberg, L. I. (1985). LEAP: A learning apprentice for VLSI design. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 573-580). Los Angeles, CA: Morgan Kaufmann.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 103-130.

Nilsson, N. J. (1965). *Learning machines.* New York: McGraw-Hill.

Nilsson, N. J. (1969). Searching problem-solving and game-playing trees for minimal cost solutions. *Information processing 68: Proceedings of the IFIP congress 1968* (pp. 1556-1562). Amsterdam: North-Holland Publishing Company.

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving.* Reading, Ma: Addison-Wesley.

Peng, J., & Williams, J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive Behaviour, 4*, 323-334.

Pomerleau, D. A. (1991). Rapidly adapting artificial neural networks for autonomous navigation. In Lippman, Moody & Touretzky (Eds.), *Advances in Neural Information Processing Systems.* San Mateo, CA: Morgan Kaufmann.

Puterman, M. L. (1994). *Markov decision processes : Discrete stochastic dynamic programming.* New York: John Wiley & Sons.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* Morgan Kaufmann.

Rich, E., & Knight, K. (1991). *Artificial intelligence.* McGraw-Hill.

Rosenbloom, P. (1982). A world-championship-level Othello program. *Artificial Intelligence, 19*, 279-320.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing.* Cambridge, MA: MIT Press.

Russell, S. J., & Norvig, P. (1995). *Artificial intelligence: A modern approach.* Englewood Cliffs, NJ: Prentice-Hall.

Sammut, C, Hurst, S, Kedzier, D, & Michie, D (1992). Learning to fly. *Machine Learning: Proceedings of the Ninth International Conference* (pp. 385-393). San Mateo, CA: Morgan Kaufmann.

Samuel, A. (1963). Some studies in machine learning using the game of Checkers. In Feigenbaum & Feldman (Eds.), *Computers and Thought.* New York: McGraw-Hill.

Samuel, A. (1967). Some studies in machine learning using the game of Checkers II: Recent progress. *IBM Journal of Research and Development, 11*, 601-617.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. *Machine Learning: Proceedings of the Tenth International Conference* (pp. 298-305). Amherst, MA: Morgan Kaufmann.

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning, 8*, 323-339.

Singh, S. P. (1994). *Learning to solve Markovian decision processes.* Doctoral dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA.

Skinner, B.F. (1938). *The behavior of organisms: An experimental analysis.* New York: D. Appleton Century.

Slagle, J.R., & Dixon, J.K. (1969). Experiments with some programs that search game trees. *J. ACM, 16*, 189-207.

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

Sutton, R. S. (1990). Integrated architectures for learning, planning and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216-224). Austin, TX: Morgan Kaufmann.

Sutton, R. S. (1991). Planning by incremental dynamic programming. *Machine Learning: Proceedings of the Eighth International Workshop* (pp. 353-357). Evanston, IL: Morgan Kaufmann.

Tesauro, G. (1995). Temporal different learning and TD-Gammon. *Communications of the ACM, 38*, 58-68.

Towell, G., Shavlik, J., & Noordewier, M. (1990). Refinement of approximate domain theories by knowledge-based neural networks. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 861-866). Boston, MA: Morgan Kaufmann.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and Q-Learning. *Machine Learning, 16*, 185-202.

Utgoff, P. E., & Clouse, J. A. (1991). Two kinds of training information for evaluation function learning. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 596-600). Anaheim, CA: MIT Press.

Utgoff, P. E. (1994). *An improved algorithm for incremental induction of decision trees*, (Technical Report 94-07), Amherst, MA: University of Massachusetts, Department of Computer Science.

Waterman, D. A. (1970). Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence, 1*, 121-170.

Watkins, C.J.C.H. (1989). *Learning with delayed rewards.* Doctoral dissertation, Psychology Department, Cambridge University.

Watkins, C.J.C.H., & Dayan, P. (1992). Q-Learning. *Machine Learning, 8*, 279-292.

Widrow, B., & Smith, F. W. (1964). Pattern-recognizing control systems. In Tou & Wilcox (Eds.), *Computer and Information Sciences Proceedings.* Washington, D.C: Spartan Books.