TEMPORAL CREDIT ASSIGNMENT IN REINFORCEMENT LEARNING

A Dissertation Presented

By

Richard S. Sutton

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

DOCTOR OF PHILOSOPHY

February 1984

Department of Computer and Information Science

© Richard S. Sutton

All Rights Reserved

This research was supported in part by the Air Force Office of Scientific Research, contract numbers AFOSR F33615-80-C-1088, AFOSR F33615-83-C-1078, and AFOSR F33615-77-C-1191.

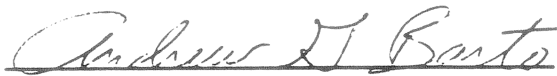
TEMPORAL CREDIT ASSIGNMENT IN REINFORCEMENT LEARNING

A Dissertation Presented

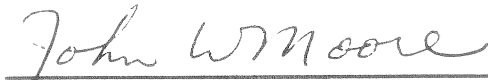
By

Richard S. Sutton

Approved as to style and content:

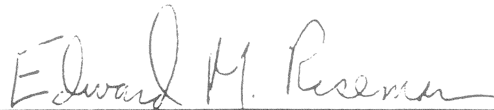


Andrew G. Barto, Chairperson of Committee



John W. Moore, Member

Michael A. Arbib, Member



Edward M. Riseman, Chairperson
Dept. of Computer and Information Science

Dedicated to Lucia and Bill

Acknowledgements

This research was carried out in close collaboration with Andy Barto over the last five and a half years. I take this opportunity to thank him for his guidance, scholarship, criticism, and friendship. Much of this research was also carried out in collaboration with Chuck Anderson. In particular, Chuck produced the software used in the pole-balancing experiment described in Chapter VI, and performed parts of that experiment. Contributions were also made by John Moore, Harry Klopf, Steve Epstein, Martha Steenstrup, P. Anandan, and John Holland. I thank Don Geman and Joe Horowitz of the Statistical Consulting Center for assistance in mathematical analysis.

To the extent that the text of this disseration is clear and readable, it is due to the guidance, criticism and editing of Andy Barto and John Moore. Michael Arbib and Oliver Selfridge also read drafts of this dissertation and recommended revisions that have been incorporated into the final dissertation.

Additional thanks are due to Harry Klopf, the AFOSR, Andy Barto, Bill Kilmer, and Nico Spinnelli for providing the opportunity for me to pursue this research, and to Michael Arbib, Sue Parker, and Ed Riseman for enhancing the environment in which it was pursued. I thank Harry Klopf for initially suggesting that I come to the UMass/Amherst COINS Dept. to pursue research in the area of learning systems.

For contributing software tools and assistance I thank Andy Cromarty, Alan Morse, Don House, Frank Glazer, Lenny Wesley, and the COINS RCF Staff.

For any errors or naïveté I alone am responsible.

Richard S. Sutton

January 3, 1984

ABSTRACT

Temporal Credit Assignment in Reinforcement Learning

February, 1984

Richard S. Sutton, B.A., Stanford University

M.S., Ph.D., University of Massachusetts

Directed by: Andrew G. Barto

This dissertation describes computational experiments comparing the performance of a range of reinforcement-learning algorithms. The experiments are designed to focus on aspects of the credit-assignment problem having to do with determining *when* the behavior that deserves credit occurred. The issues of knowledge representation involved in developing new features or refining existing ones are not addressed.

The algorithms considered include some from learning automata theory, mathematical learning theory, early "cybernetic" approaches to learning, Samuel's checker-playing program, Michie and Chambers's "Boxes" system, and a number of new algorithms. The tasks were selected so as to involve, first in isolation and then in combination, the issues of misleading generalizations, delayed reinforcement, unbalanced reinforcement, and secondary reinforcement. The tasks range from simple, abstract "two-armed bandit" tasks to a physically realistic pole-balancing

task.

The results indicate several areas where the algorithms presented here perform substantially better than those previously studied. An unbalanced distribution of reinforcement, misleading generalizations, and delayed reinforcement can greatly retard learning and in some cases even make it counterproductive. Performance can be substantially improved in the presence of these common problems through the use of mechanisms of reinforcement comparison and secondary reinforcement. We present a new algorithm similar to the "learning-by-generalization" algorithm used for altering the static evaluation function in Samuel's checker-playing program. Simulation experiments indicate that the new algorithm performs better than a version of Samuel's algorithm suitably modified for reinforcement-learning tasks. Theoretical analysis in terms of an "ideal reinforcement signal" sheds light on the relationship between these two algorithms and other temporal credit-assignment algorithms.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS v
ABSTRACT vi
LIST OF TABLES xi
LIST OF ILLUSTRATIONS xii

Chapter

I. INTRODUCTION 1

Artificial Intelligence 3
Reinforcement Learning 4
The Critic and Heuristic Reinforcement 6
Secondary Reinforcement 8
Simplifications and Extensibility 9
Methodology 10
Overview by Chapters 11

II. EXPERIMENTS WITH NONASSOCIATIVE TASKS 13

Reinforcement-Comparison Algorithms 14
The Independence-of-Path Assumption 16
Tasks 17
Algorithms 20
Results 23
Discussion 33
Absolute Expediency of a Reinforcement-Comparison Algorithm 38
Extensions 40
Conclusions 41

III. ASSOCIATIVE LEARNING 43

Approaches to Associative Learning 44
Independent-Step Associative Learning 53
Tasks 54
Algorithms 57
Results 63
Discussion 80
General Discussion 89

IV. DELAYED REINFORCEMENT 92

Eligibility Traces 93
Reinforcement Comparison Under Delayed Reinforcement . . . 97
Experiment 1: The Effect of Delay 100
Experiment 2: Sooner Is Better 108
Experiment 3: Comparison of Algorithms on Nonassociative Tasks 112
Experiment 4: Associative-Delay Asymmetry 116
Conclusions 118

V. SECONDARY REINFORCEMENT ALGORITHMS 121

The Ideal Reinforcement Signal 122
The Adaptive Heuristic Critic Algorithm 126
Time-Blind Tasks 134
Time-until-Failure Tasks 139
Time-until-Success Tasks 140
Samuel's "Learning-by-Generalization" Algorithm 141
Witten's Adaptive Controller 145
Equivalent Expressions for the AHC Algorithm 147
Conclusions 150

VI. EXPERIMENTS INVOLVING SECONDARY REINFORCEMENT . 151

Illustrations of the Behavior of the AHC Algorithm 152
Sutton and Barto's Classical Conditioning Model 164
Experiment 1: Reinforcement Anticipation 166
Experiment 2: An "Easy" Action Sequence 172
Experiment 3: Misleading Generalizations 177
Experiment 4: Pole-Balancing 182
Summary 189

VII. CLOSING 192
Reinforcement Comparison 192
Ideal Reinforcement Signal 194
The AHC Algorithm and Samuel's Checker-Playing Program . 195
Limitations and Future Work 196
Applications 197
Reinforcement Learning 199
REFERENCES 200

LIST OF TABLES

1. Summary of Nonassociative Tasks 18
2. Nonassociative Learning Algorithms 21
3. Associative Learning Tasks 55
4. Associative Learning Algorithms 59
5. Statistical Significance Results of Chapter III 79
6. Learning Algorithms of Chapter IV 98
7. Learning Algorithms of Chapter VI 168
8. Details of Cart-Pole Simulation 186

LIST OF ILLUSTRATIONS

1. Information Flow Among Components of a Learning Process 7
2. Algorithm Performance on Task 1 of Chapter II 25
3. Algorithm Performance on Task 2 of Chapter II 26
4. Algorithm Performance on Task 3 of Chapter II 27
5. Algorithm Performance on Task 4 of Chapter II 28
6. Algorithm Performance on Task 5 of Chapter II 29
7. Algorithm Performance on Task 6 of Chapter II 30
8. Summary of Algorithm Performance on all Tasks of Chapter II . . 31
9. Algorithm Performance on Task 1 of Chapter III 65
10. Algorithm Performance on Task 2 of Chapter III 66
11. Algorithm Performance on Task 3 of Chapter III 67
12. Algorithm Performance on Task 4 of Chapter III 68
13. Algorithm Performance on Task 5 of Chapter III 69
14. Algorithm Performance on Task 6 of Chapter III 70
15. Algorithm Performance on Task 7 of Chapter III 71
16. Algorithm Performance on Task 8 of Chapter III 72
17. Algorithm Performance on Task 9 of Chapter III 73
18. Algorithm Performance on Task 10 of Chapter III 74
19. Algorithm Performance on Task 11 of Chapter III 75
20. Algorithm Performance on Task 12 of Chapter III 76
21. Summary of Algorithm Performance on Tasks 1–10 of Chapter III . 78
22. Effect of Delayed Reinforcement on the Performance of Algorithms Without Eligibility Traces 103
23. Effect of Delayed Reinforcement on the Performance of Algorithms with Short Eligibility Traces 104
24. Effect of Delayed Reinforcement on the Performance of Algorithms with Moderate Length Eligibility Traces 105
25. Effect of Delayed Reinforcement on the Performance of Algorithms with Long Eligibility Traces 106
26. Interaction of Eligibility Trace Length and Performance on Delayed-Reinforcement Tasks 107
27. Comparison of Algorithm Performance with and without Delay Asymmetry 110
28. Algorithm Performance on *High* Task 113

29. Algorithm Performance on *Low* Task 114
30. Algorithm Performance on *Middle* Task 115
31. Summary of Algorithm Performance on a Task with Associative-Delay Asymmetry 118
32. State-Transition Structure of two Environments that are Problematic for Definitions of the Ideal Reinforcement Signal . . . 129
33. An Environment that is very Different as a Time-Blind and as a Non-Time-Blind Task 136
34. Behavior of the AHC and RC Algorithms when Presented with a Brief Stimulus Followed by Primary Reinforcement 154
35. Behavior of the AHC Algorithm when Presented with a Temporal Sequence of Stimuli Followed by Primary Reinforcement . 156
36. Behavior of the RC Algorithm when Presented with a Temporal Sequence of Stimuli Followed by Primary Reinforcement 158
37. Behavior of the AHC Algorithm when Presented with a Stimulus and Primary Reinforcement in a Variety of Temporal Configurations 159
38. Behavior of the AHC Algorithm when Presented with two Stimuli and Primary Reinforcement in a Variety of Temporal Configurations 161
39. Behavior of the AHC Algorithm with $\gamma = .98$ when Presented with a Sequence of Stimuli Followed by Primary Reinforcement . . 163
40. State-Transition Structure of the Environment used in Experiment 1 of Chapter VI 167
41. Algorithm Performance on Experiment 1 of Chapter VI 170
42. State-Transition Structure of the Environment used in Experiment 2 of Chapter VI 173
43. Algorithm Performance on Experiment 2 of Chapter VI 178
44. State-Transition Structure of the Environment used in Experiment 3 of Chapter VI 179
45. Algorithm Performance on Experiment 3 of Chapter VI 180
46. The Cart-Pole System to be Controlled in Experiment 4 of Chapter VI 183
47. Average Performance of 3 Best Algorithms on Pole-Balancing Task, Logarithmic Scale 188
48. Average Performance of 3 Best Algorithms on Pole-Balancing Task, Linear Scale 189
49. Individual Runs of 3 Best Algorithms on Pole-Balancing Task 190

CHAPTER I

INTRODUCTION

The credit-assignment problem for a complex learning system (Minsky, 1961) is the problem of properly assigning credit or blame for overall outcomes to each of the learning system's internal decisions that contributed to those outcomes. In many cases the dependence of outcomes on internal decisions is mediated by a sequence of actions generated by the learning system. That is, internal decisions affect which actions are taken, and then the actions, not the internal decisions, directly influence outcomes. In these cases it is sometimes useful to decompose the credit-assignment problem into two subproblems: 1) the assignment of credit for outcomes to actions, and 2) the assignment of credit for actions to internal decisions. The first subproblem involves determining *when* the actions that deserve credit were taken, and the second involves assigning credit to the internal structure of actions. Accordingly, the first subproblem is called the *temporal credit-assignment problem*, and the second the *structural credit-assignment problem*.

For example, consider the difficulties a learning system faces in assigning credit for the outcome (win, loss, or draw) of a game of chess. The outcome depends on the moves selected, but the moves typically depend on a multitude of internal decisions made by the learning system. The temporal credit-assignment problem is to assign credit to moves for the game's outcome, by determining precisely when the position became worse and when better. The structural credit-assignment

problem is to determine which internal decisions are responsible for the selection of each move, and thereby convert the credit assigned to moves into credit assigned to internal decisions.

The original complete name given to the credit-assignment problem by Minsky (1961) was "the basic credit-assignment problem for complex *reinforcement learning systems*" (emphasis added). Although the current use of the term credit-assignment is not restricted to reinforcement learning (e.g., Dietterich et. al., 1982), the complete name nevertheless reveals the close relationship between them. For the purposes of this dissertation, a *learning system* is defined as a system that uses information gained during one interaction with its *environment* to improve its performance in subsequent interactions (after Smith, 1980). A *reinforcement-learning system* is a learning system that learns under the influence of reinforcement, where *reinforcement* is feedback from the environment that assigns credit to the learning system's actions but does not assign credit to their internal structure or indicate what actions would have been better.

Supervised learning pattern classification (see e.g., Duda and Hart, 1973), for example, is not reinforcement learning because the environment, in this case called a "teacher," indicates what actions should have been taken and thus implicitly assigns credit to particular internal decisions. The hypothetical chess-player discussed above, on the other hand, would be engaged in reinforcement learning because the outcome of a game only evaluates actions made; it does not indicate which actions should have been made, and it provides no information at all about the decision processes internal to the learning system.

This dissertation describes a series of experiments focused on the temporal credit-assignment problem in reinforcement learning. A range of algorithms from the artificial intelligence and learning automata theory literature, as well as several new algorithms, are systematically compared. Whenever possible, the issues re-

garding structural credit-assignment are postponed for future work. The separation of temporal and structural credit assignment enables complex issues in temporal credit assignment to be studied in relatively simple reinforcement-learning tasks.

Artificial Intelligence

The earliest studies of learning in artificial intelligence (AI) occurred in the 1950's (e.g., Farley and Clark, 1954; Minsky, 1954; Selfridge, 1956, 1959; Rosenblatt, 1957). By the end of the decade enough experience had been accumulated with complex learning tasks (e.g., Newell, 1955; Samuel, 1959; Friedberg, 1958; Friedberg et. al., 1959) to identify some of the fundamental difficulties in constructing effective learning systems (e.g., see Minsky and Selfridge, 1961; Minsky, 1961). Foremost among these was the credit-assignment problem.

Most learning programs during these early years of AI were capable of only rudimentary forms of credit assignment and thus could address only relatively simple problems in which credit assignment was very easy. The algorithms developed by Farley and Clark (1954) and Rosenblatt (1957), for example, assigned credit equally to each decision leading to an overall outcome. Samuel, on the other hand, directly addressed many important issues in credit assignment and used more sophisticated algorithms in his celebrated checker-playing program (Samuel, 1959, 1967).

As the difficulties in attaining effective learning systems became apparent, interest in learning in AI waned. From the mid 1960's to the end of that decade relatively little work was done on learning or credit assignment within AI. AI's shift away from learning was solidified by the elucidation of some theoretical limitations of the simplest learning machines (Minsky and Papert, 1969) and further disappointing results (e.g., Fogel, Owens, and Walsh, 1966). The early AI research

into learning and adaptive systems was subsequently pursued more in the fields of pattern classification (e.g., see Duda and Hart, 1973) and function optimization (e.g., see McMurtry, 1970; Jarvis, 1975) than in AI.

The 1970's saw a gradual renewal of interest in learning. Several influential AI systems appeared during this decade that incorporated learning at a high level (e.g., Winston, 1970; Waterman, 1970; Sussman, 1973; Buchanan et. al., 1976; Lenat, 1976; Mitchell, 1978; Soloway, 1978). In the 1980's, this trend appears to be continuing and strengthening (e.g., Minsky, 1980; Dietterich and Michalski, 1981; Langley and Simon, 1981; Rissland and Soloway, 1981; Mitchell et. al., 1981; Carbonell, 1982; Feldman, 1981, 1982; Schank, 1983). Concomitantly, there is renewed interest in, and growing appreciation of, the credit-assignment problem. Dietterich et. al. (1982), for example, notes the increasing interest in learning, and interprets the learning algorithms of several AI systems in terms of their techniques for solving the credit-assignment problem.

Reinforcement Learning

The term reinforcement learning seems to have come into use in AI and the engineering disciplines through early work by Minsky (1954, 1961; see also Minsky and Selfridge, 1961). Psychologists do not use this particular phrase, although the idea has plainly arisen from their study of reinforcement, and particularly of instrumental (operant) conditioning. The field of ***reinforcement learning control theory*** also uses the term (e.g., Mendel, 1966; Mendel and McClaren, 1970), and it is used in many studies of machine learning in areas related to AI.

A commonly studied special case of reinforcement learning is that in which the learning system's task is to select a ***single*** optimal action rather than to associate different actions with different stimuli. In such ***nonassociative learning problems***

the reinforcement is usually the only input the learning system receives from its environment. Nonassociative reinforcement learning has been studied as function optimization (e.g., McMurtry, 1970; Holland, 1975), which includes the study of hillclimbing algorithms (e.g., Howland, Minsky and Selfridge, 1960), as learning automata theory (e.g., Narendra and Thathachar, 1974), and in regard to the two-armed bandit problem (e.g., Cover and Hellman, 1970). Although several issues in nonassociative reinforcement learning are considered in detail in Chapter II, the primary focus in this dissertation is *associative reinforcement learning.* In associative reinforcement learning tasks, different actions may be optimal in response to different stimuli. Consequently, the learning system receives stimuli other than reinforcement, and stimulus-action associations must be learned.

The work of Klopf (1972, 1982) has emphasized that associative reinforcement learning is different from both nonassociative reinforcement learning and other types of associative learning (e.g., supervised learning pattern classification). Associative reinforcement learning has not received as much attention from cyberneticians and AI researchers as has these other types of learning, yet it is an obvious way to improve the performance of a problem-solving system; its essence is the "caching" of search results in an associative memory so that future search is converted into simple memory access. The importance of this process is appreciated by AI researchers (see, e.g., Lenat, Hayes-Roth, and Klahr, 1979), but its use is not widespread.

Some of the earliest AI work concerned associative reinforcement learning (Farley and Clark, 1954; Minsky, 1954; Minsky and Selfridge, 1961). However, interest in the field quickly shifted from associative reinforcement learning to other associative-learning problems such as pattern classification. Modern AI research on learning systems has also, for the most part, declined to consider associative reinforcement learning and its credit-assignment problems.

Outside of AI, algorithms for associative reinforcement learning have been considered in psychology's mathematical learning theory (e.g., Bush and Mosteller, 1955; Bush and Estes, 1959), in learning automata theory (e.g., see Narendra and Thathachar, 1974), and in reinforcement learning control theory (Mendel and McClaren, 1970). In the latter two cases, however, the primary focus has been on nonassociative reinforcement learning, with associative reinforcement learning treated as multiple independent instances of nonassociative reinforcement learning. A similar approach has been taken by Jarvis (1970), in which separate pattern recognition and nonassociative reinforcement-learning algorithms are combined.

The Critic and Heuristic Reinforcement

It is common in AI to divide a learning system into components including a performance element, a knowledge base, and a learning element (Buchanan et.al., 1978; Dietterich et.al., 1982). Figure 1 shows such a division for the learning systems examined in this dissertation. All lines of communication among the learning system's major components and between the learning system and its environment are shown. The learning system receives *stimuli* and a *primary reinforcement signal* from the environment, and in return generates actions. The *performance element* is the component of the learning system responsible for selecting actions as a function of stimuli, where the selections or selection probabilities at a given time depend on the state of the *knowledge base*. The *learning element* is responsible for making all changes in the knowledge base. Breaking slightly with earlier usage (Buchanan et.al., 1978), the term *critic* is used here to refer to the component of the learning system responsible for converting primary reinforcement into a higher quality reinforcement signal. This second reinforcement signal is said to be based on heuristics, either given a priori or learned, and is accordingly called the *heuristic reinforcement signal*.

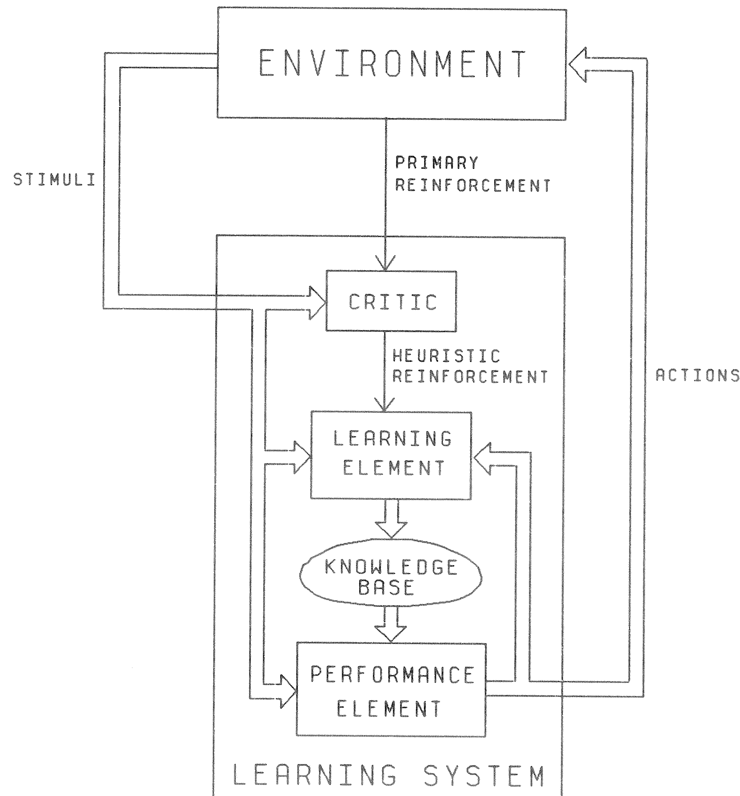


Figure 1. Information Flow Among Components of a Learning Process. Single width lines indicate pathways restricted to transmitting scalars. The critic component transforms primary reinforcement to heuristic reinforcement and receives information from no other component of the learning system.

Since the critic's only output is a reinforcement signal, it can contribute to the learning process only by aiding temporal credit assignment, not structural credit assignment. By generating a high (low) reinforcement value at a certain time, the critic indicates that credit (blame) should be assigned to the action chosen at that time, but the critic has no means to indicate what action would have been better, what aspects of the action should have been different, or how the internal decision-making processes of the performance element should be changed. All of this structural credit assignment is assumed to be performed by the learning

element, using the heuristic reinforcement provided by the critic. The research reported in this dissertation is primarily concerned with temporal credit assignment performed by a critic, i.e., by converting primary to heuristic reinforcement with the aid of past experience and stimulus information.

Secondary Reinforcement

One way in which a critic can generate a heuristic reinforcement signal that is more useful than the primary reinforcement signal is through *secondary reinforcement*. If particular stimuli are regularly followed by high reinforcement, then a good critic should assign credit to behavior occurring near the time of occurrence of those stimuli, rather than to behavior occurring near the time of the high reinforcement. Credit should be assigned at the earlier time because that was when the critic was first informed that the high reinforcement was coming, and thus the most likely time of the action that caused it. Credit delivered by stimuli that have acquired reinforcing properties through this kind of association with primary reinforcement is called *secondary reinforcement*; the stimuli themselves are called *secondary reinforcers*. Stimuli can also become secondary reinforcers by association with previously established secondary reinforcers rather than with primary reinforcement. Thus, secondary reinforcement can be "chained" backwards. Minsky (1961) has discussed the addition of secondary-reinforcement mechanisms to reinforcement-learning systems.

Although secondary reinforcement is one of the most important ways in which heuristic reinforcement can be an improvement over primary reinforcement, it is excluded from consideration in the early chapters of this dissertation. Chapters II, III, and IV concentrate instead on other temporal credit-assignment issues and on *reinforcement-comparison mechanisms* (see Chapter II), the second major class of

mechanisms studied here for constructing helpful heuristic reinforcement signals.

Because information about the actions selected is not used in temporal credit assignment, or at least it is not in any of the cases considered here, Figure 1 shows the critic as not receiving any information from the performance element. A temporal credit-assignment algorithm in a sense assigns credit to *times* and thus indirectly to the actions made at those times, rather than to the actions themselves.

Simplifications and Extensibility

The research strategy taken here is predicated on the assumption that the problems of temporal credit assignment are in the main separable from those of structural credit assignment. If this is correct, then the results and new algorithms presented here should be extensible to tasks and learning systems that are much more sophisticated with respect to knowledge representation and structural credit assignment.

AI is an experimental science, yet the complexity of its programs and problem domains often makes the interpretation of results very difficult. Programs often contain so many components and parameters that limitations on computer time and the sheer number of possibilities make it impossible to experimentally evaluate how each contributes to performance. Although the research reported here does not fully escape these problems, they are greatly eased by simplifying all aspects of the tasks and learning algorithms not directly relevant to temporal credit assignment. By virtue of these simplifications, temporal credit-assignment issues are explored much more thoroughly than would otherwise have been possible, albeit at the cost of neglecting other issues, most notably those of knowledge representation and structural credit assignment.

The most important simplifications made here are that, in all tasks considered, 1) the learning system chooses between only two actions at each time, and 2) the correct stimulus-action mapping can be implemented by a performance element and knowledge base consisting of a linear-threshold operation and a single real-valued vector. Since enormously more complex decision-making processes are commonly studied in AI, these are drastic simplifications. However, the complexity of the decision-making mechanism enters into the credit-assignment process only *after* credit has been assigned to the actions taken at each time, and thus it affects structural but not temporal credit assignment. Temporal credit is assigned independently of the actions selected and of the action-selection process. Assigning temporal credit using secondary reinforcement, for example, is done purely on the basis of observing stimuli and their temporal relationship to primary reinforcement; knowledge of the actions selected or of how they were selected is not needed and not used. In the case of playing chess, discussed earlier, temporal credit assignment consists of determining exactly when one's position has become worse and when better. The difficulty of this judgment is not affected by the complexity of the process used to select moves; it can be made purely by inspection of board positions. Further, a good temporal credit-assignment algorithm for one move-selection process should also work well with any other.

Methodology

The methodology used throughout most of this dissertation relies strongly on computer simulation. Through series of simulation experiments, the relative performance (usually, the rate of learning) of a variety of algorithms on specific tasks is determined. Wherever possible, algorithms proposed by other researchers are included, allowing the demonstration of advances over existing methods.

The research reported here progresses incrementally from very simple and constrained tasks to more complex and general one with successive chapters. Each increment introduces a characteristic set of problems and issues, and suggests a new set of experiments and algorithms for investigation, some of which are explored. In addition, algorithms and tasks analogous to those of preceding chapters, suitably generalized, are examined to extend and check the generality of earlier results.

Overview by Chapters

Chapter II describes experiments with discrete-action nonassociative reinforcement learning. By excluding the associative aspect, the experiments of this chapter focus on reinforcement learning in one of its most elemental forms. The algorithms compared in this chapter include several well-studied learning automata algorithms. The experiments of this chapter introduce the problem of *unbalanced reinforcement* and mechanisms of *reinforcement comparison* that help solve it. Results are presented that strongly suggest the superior performance (learning rate) of algorithms using reinforcement-comparison mechanisms. In Chapter III reinforcement-comparison mechanisms are generalized from nonassociative to associative reinforcement learning. The results of this chapter's experiments show that some intuitively satisfying algorithms have serious difficulties when compared to other algorithms. The experiments of Chapter IV deal with the deleterious effects on learning of delayed reinforcement. The results confirm those of preceding chapters and establish a baseline performance level with which to compare that of secondary reinforcement algorithms. Chapter V discusses the design of an *adaptive heuristic critic* (AHC) algorithm for implementing secondary reinforcement, and relates it to the temporal credit-assignment algorithms used by Samuel (1959, 1967) and Witten (1977). In Chapter VI experiments are described illustrating the behavior of the AHC algorithm and comparing its performance with that of

other algorithms. One of the tasks considered in Chapter VI is that of learning to balance a pole under conditions that create a difficult temporal credit-assignment problem. This task is due to Michie and Chambers (1968a,b), whose "Boxes" algorithm is one of those with which performance on this task is compared. Chapter VII contains concluding remarks.

CHAPTER II

EXPERIMENTS WITH NONASSOCIATIVE TASKS

This first series of experiments concerns nonassociative tasks in which the only signal the learning system receives from its environment is a scalar primary reinforcement signal. Such tasks are nonassociative because there is no non-reinforcing stimuli with which to associate actions. The learning system must find and consistently choose an action so as to maximize the expected value of the primary reinforcement signal.

The simplest reinforcement-learning tasks are nonassociative, and the tasks considered in this chapter are particularly simple nonassociative tasks. Nevertheless, temporal credit-assignment issues arise in this chapter's tasks that also affect performance on more complex tasks, including those considered in subsequent chapters. The issues can be investigated much more easily and thoroughly in the simple tasks than in the complex ones. Subsequent chapters concern tasks of greater complexity and generality, but rely on the results obtained with simpler tasks in the present chapter.

The experiments described in this chapter were designed to compare the convergence rates of a variety of algorithms across a variety of nonassociative tasks. Although ideally one would like to determine analytically the relative convergence rates of the algorithms, in practice this is very difficult. Whereas a large number of disparate algorithms and tasks can be considered experimentally, mathematical

results concerning convergence rates are difficult to obtain even in simple cases. One mathematical result is presented relating to the convergence (but not rate of convergence) of one of the new algorithms presented here.

Reinforcement-Comparison Algorithms

There are two broad traditions of research into two types of nonassociative reinforcement-learning problems. One is that of function optimization (including the study of hillclimbing algorithms) (see e.g., McMurtry, 1970, or Jarvis, 1975), and the other is that of learning automata theory (LAT) (e.g., Narendra and Thathachar, 1974; Lakshmivarahan, 1981) and mathematical learning theory (MLT) (e.g., Bush and Mosteller, 1955; Bush and Estes, 1959; Luce, Bush and Galanter, 1963, 1965; Atkinson, Bower and Crothers, 1965). The LAT and MLT researchers are primarily interested in problems with discrete action spaces in which the learning system chooses one out of two, or one out of n, actions, whereas those studying function optimization are primarily interested in continuous action spaces, such as the parameter space of a control system. In addition, LAT and MLT researchers are most interested in problems that have discrete outcomes, such as binary-reinforcement, or success/failure, problems, while those who study function optimization are most interested in problems with continuous-valued performance measures. These differences have led to differences in the way problems are formulated, and ultimately to differences in algorithms and how they are evaluated. The major type of problem studied in LAT and MLT has also been studied as the "n-armed bandit problem" (e.g., Bradt, et al., 1956; Cover and Hellman, 1970).

Most research into associative reinforcement learning has concerned algorithms that either evolved from, or are very similar to, those studied in LAT and MLT. Accordingly, this tradition of research into nonassociative reinforcement-learning

problems is focused on in this chapter's experiments. The algorithms developed within this tradition are global-search algorithms that cannot become trapped on local optima. Unlike hillclimbing algorithms, for example, they do not search by trying actions that are near previous actions. Actions are not related to one another in terms of nearness or similarity, or in any other sense. Since they are not local, gradient methods, these algorithms typically do not incorporate explicit comparisons of current reinforcement levels with past reinforcement levels. Within the framework used in the LAT-MLT tradition, it is not useful to estimate the change in reinforcement as a function of change in action (since all changes in actions are the same). It may nevertheless be useful to compare the current reinforcement level with past reinforcement levels. Algorithms that do this are called *reinforcement-comparison algorithms*. As is discussed below, almost all LAT and MLT research seems to have been restricted to algorithms that are not reinforcement-comparison algorithms (called non-reinforcement-comparison algorithms).

A task is called an *unbalanced-reinforcement task* if its primary reinforcement signal is biased towards a predominance of either positive values (indicating credit should be assigned to the learning system's behavior) or negative values (indicating blame). Tasks in which the distribution of reinforcement signal values is balanced around the neutral value zero are called *balanced-reinforcement tasks*. One might expect a reinforcement-comparison mechanism to provide an advantage on unbalanced-reinforcement tasks. By comparing current reinforcement with past reinforcement, a reinforcement-comparison mechanism can produce a new (heuristic) reinforcement signal which is balanced or nearly balanced. Removing imbalances in this way may aid learning. The experiments described in this chapter were designed to test this hypothesis.

The Independence-of-Path Assumption

The failure to consider reinforcement-comparison algorithms in MLT can be traced to the "independence-of-path assumption" that practically all researchers in this field adopted. The ***independence-of-path assumption*** is the assumption that the only memory of past experience a learning system retains is contained in the probabilities of selecting each action. In other words, these action probabilities encode the entire state of the learning system. Its subsequent behavior is completely ***independent of the path*** (i.e., sequence of events) by which the action probabilities reached their current values.

The independence-of-path assumption precludes the consideration of reinforcement-comparison algorithms. A reinforcement-comparison algorithm by definition retains memory about past levels of reinforcement and uses it to guide changes in its action probabilities. Thus a reinforcement-comparison algorithm's current state includes its memory of past reinforcement levels in addition to the action probabilities, and it cannot meet the independence-of-path assumption.

As learning automata were initially defined (Tsetlin, 1961, 1973), they included the possibility of reinforcement-comparison algorithms. However, possibly due to the strong influence MLT has had on LAT, most contemporary LAT is restricted to algorithms that meet the independence-of-path assumption. In brief, a stochastic learning automaton is an automaton whose next state is determined by a probability distribution over its states, where the probability distribution is modifiable by a learning process. If the states of the automaton are placed in one-to-one correspondence with its actions, as is often done, then the automaton retains no memory except for that of its previous action and its current probability distribution. Since knowledge of the previous action is used to update the probability distribution, and not to select actions, this restriction is identical to that of the independence-of-path assumption.

Contemporary LAT research seems to be restricted to the case of a one-to-one correspondence between actions and states. For example, Narendra and Thathachar in a review article (Narendra and Thathachar, 1974) define learning automata in the general case, but then go on to consider only the restricted class of automata. In his book Lakshmivarahan (1981), another researcher in this area, simply includes the restriction in his definitions. I have recently become aware of several algorithms discussed by Mars and Poppelbaum (1981) which *are* reinforcement-comparison algorithms. The relationship between these algorithms and those discussed here has not yet been investigated.

MLT and LAT have demonstrated that non-reinforcement-comparison algorithms are capable of eventually solving nonassociative reinforcement-learning problems. Several forms of asymptotic near optimality — near-optimal performance as time goes to infinity — have been shown for the non-reinforcement-comparison algorithms considered in these fields. However, in practice, rate of convergence is of critical importance to the usefulness of an algorithm, and those that have been proven to to be near optimal have tended to be very slow. This chapter describes a series of computer simulations of 10 learning algorithms, some that are reinforcement-comparison algorithms, and some that are not, applied to 6 nonassociative reinforcement-learning tasks. These experiments were designed to compare the convergence rates of the algorithms across tasks.

Tasks

The 6 learning tasks are summarized in Table 1. There are 3 each of 2 types of tasks, called *binary-reinforcement tasks* and *continuous-reinforcement tasks*. In

Table 1. Summary of Nonassociative Tasks.

Task Number	Reinforcement Type	r range	r mean Action 1	r mean Action 0	Relevant Algorithms
1	Binary	$\{1, -1\}$	.9	.8	1-9
2	Binary	$\{1, -1\}$	.2	.1	1-9
3	Binary	$\{1, -1\}$	.55	.45	1-9
4	Continuous	$\Re$	.9	.8	4-10
5	Continuous	$\Re$	$-.8$	$-.9$	4-9
6	Continuous	$\Re$	.05	$-.05$	4-9

binary-reinforcement tasks each interaction of the learning system with the environment has one of two outcomes. The goal of the learning system is to maximize the number of interactions that result in one outcome, called *success*, and minimize the number that result in the other outcome, called *failure*. The learning system is made aware of the outcome of its action $y[t]$ taken at time t by the reinforcement $r[t+1]$ it receives at time $t+1$. If the outcome is success, $r[t+1]$ has the value $+1$, and if the outcome is failure, $r[t+1]$ has the value -1.

In continuous-reinforcement tasks each outcome is a real-valued reinforcement. For example, the reinforcement $r[t+1]$ corresponding to the action $y[t]$ might take on values in the real interval $[-1, +1]$. In continuous-reinforcement tasks the goal of learning is to maximize the expected value of the reinforcement.

All 6 tasks are binary-action tasks: the learning system chooses one of two actions, Action 0 ($y[t] = 0$) or Action 1 ($y[t] = 1$).

For binary-reinforcement tasks, the environment is fully characterized by two

conditional probabilities. One is the probability that $r[t+1] = +1$ given that $y[t] = 0$, and the other is the probability that $r[t+1] = +1$ given that $y[t] = 1$. The success probabilities conditional on each action for the three binary-reinforcement tasks are given in Columns 4 and 5 of the first three rows of Table 1.

For simplicity the tasks were constructed such that Action 1 is the better action on all tasks*. On Tasks 1–3 this was achieved by selecting a higher success probability conditional on Action 1 than that conditional on Action 0. These tasks include one on which both success probabilities are high (Task 1), one on which both success probabilities are low (Task 2), and one on which one of the success probabilities is greater than $\frac{1}{2}$ and the other less than $\frac{1}{2}$ (Task 3). Tasks in which the distribution of positive and negative reinforcement values is not balanced are called ***unbalanced-reinforcement tasks.*** Tasks 1 and 2 (and Tasks 4 and 5, discussed below) are unbalanced-reinforcement tasks.

One might expect an unbalanced-reinforcement task such as Task 1 to present a special problem for learning algorithms because even the incorrect action is followed by success almost all (80%) of the time. An unbalanced-reinforcement task such as Task 2 might also be problematic because the correct action is followed by failure almost all (80%) of the time. By contrast, a balanced-reinforcement task such as Task 3 should be easy to solve, because the correct action is successful more than half the time and the incorrect action is unsuccessful more than half the time. Thus, these three tasks cover the major classes of non-trivial binary-reinforcement/binary-action tasks.

Columns 4 and 5 of the last three rows of Table 1 list the expected value of the reinforcement signal for each action on the three continuous-reinforcement tasks. For each of these tasks reinforcement is selected according to a uniform distribution

---

* Since all learning algorithms applied to these tasks are symmetrical with respect to the two actions, they can not take advantage of this uniformity.

centered at the mean indicated in Table 1 and extending 0.1 to either side. For example, if the learning system selects Action 1 at time t on Task 4, then the reinforcement at time $t+1$ is selected according to a uniform distribution over the interval from 0.8 to 1.0. The means and widths of the uniform distributions determining reinforcement were selected close together and narrow respectively so as to amplify differences between binary-reinforcement and continuous-reinforcement tasks.

The three continuous-reinforcement tasks (Tasks 4–6) were selected to cover roughly the same space of relative difficulties as the binary-reinforcement tasks. One task was constructed so that the expected value of reinforcement would be positive for both actions (Task 4), one so that the expected value of reinforcement would be negative for both actions (Task 5), and one so that the expected value of reinforcement would be positive for one action and negative for the other (Task 6). Tasks 4 and 5 are unbalanced reinforcement tasks.

Algorithms

The 10 learning algorithms are summarized in Table 2. Each algorithm selects its actions probabilistically; $\pi[t]$ denotes the probability with which Action 1 is selected at time t. Not all algorithms were applied to all tasks. The tasks to which each algorithm was applied are indicated in the last column of Table 2.

Algorithms 1–3 are well-known learning automata algorithms. They are, respectively, linear reward-inaction (L_{RI}), linear penalty-inaction (L_{PI}), and linear reward-penalty (L_{RP}). These algorithms apply different update equations for success and failure and are only applicable to the binary-reinforcement tasks.

Algorithm 10 is also a learning automaton algorithm (Mason, 1973). It was

Table 2. Nonassociative Learning Algorithms.

Algorithm	Update Rule	Relevant Tasks
1	$\pi[t+1] = \pi[t] + \begin{cases} \alpha(y[t] - \pi[t]), & \text{if } r[t+1] = 1 \\ 0, & \text{if } r[t+1] = -1 \end{cases}$	1-3
2	$\pi[t+1] = \pi[t] + \begin{cases} 0, & \text{if } r[t+1] = 1 \\ \alpha(1 - y[t] - \pi[t]), & \text{if } r[t+1] = -1 \end{cases}$	1-3
3	$\pi[t+1] = \pi[t] + \begin{cases} \alpha(y[t] - \pi[t]), & \text{if } r[t+1] = 1 \\ \alpha(1 - y[t] - \pi[t]), & \text{if } r[t+1] = -1 \end{cases}$	1-3
4	$w[t+1] = w[t] + \alpha r[t+1](y[t] - \frac{1}{2})$	1-6
5	$w[t+1] = w[t] + \alpha r[t+1](y[t] - \pi[t])$	1-6
6	$w[t+1] = w[t] + \alpha(r[t+1] - r[t])(y[t] - \frac{1}{2})$	3,4
7	$w[t+1] = w[t] + \alpha(r[t+1] - r[t])(y[t] - \pi[t])$	3,4
8	$w[t+1] = w[t] + \alpha(r[t+1] - p[t])(y[t] - \frac{1}{2})$	1-6
9	$w[t+1] = w[t] + \alpha(r[t+1] - p[t])(y[t] - \pi[t])$	1-6
10	$\pi[t+1] = \pi[t] + \alpha r[t+1](y[t] - \pi[t])$	4

Where:

$$w[0] = 0, \qquad \pi[0] = \frac{1}{2}, \qquad y[t] \in \{1, 0\}, \qquad \alpha > 0,$$

and $\pi[t]$ is the probability that $y[t] = 1$.

For Algorithms 4-9: $$y[t] = \begin{cases} 1, & \text{if } w[t] + \eta[t] > 0; \\ 0, & \text{otherwise,} \end{cases}$$

where $\eta[t]$ is a normally distributed random variable of mean 0 and standard deviation $\sigma_y = .3$.

For Algorithms 8 and 9: $p[t+1] = p[t] + \beta(r[t+1] - p[t])$, where $p[0] = r[1]$ and $\beta = 0.2$.

designed for tasks in which the environment returns a real-valued reinforcement in the interval from 0 and 1. These tasks are called S-model environments in the learning automata literature (see e.g., Narendra and Thathachar, 1974). Algorithm 10 was applied only to Task 4, because this is the only task in which the reinforcement is always between 0 and 1.*

Whereas Algorithms 1–3 and 10 directly update the probability $\pi[t]$ of choosing Action 1, Algorithms 4–9 operate by updating a modifiable parameter $w[t]$ that determines this probability according to:

$$\pi[t] = \Phi(w[t]/\sigma_y), \tag{1}$$

where $\Phi(\cdot)$ denotes the unit normal distribution function, and $\sigma_y = .3$. Computationally, Algorithms 4–9 determine their action $y[t]$ according to the sign of the sum of $w[t]$ and a random variable $\eta[t]$ selected according to a normal distribution with mean 0 and standard deviation σ_y. If the sum is greater than 0, Action 1 is chosen, otherwise Action 0 is chosen:

$$y[t] = \begin{cases} 1, & \text{if } w[t] + \eta[t] > 0; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

This manner of selecting actions yields the probability $\pi[t]$ as a function of $w[t]$ given by (1). The normally-distributed random variables used in these experiments were obtained by passing a uniformly-distributed (pseudo-)random number though the inverse of the normal-distribution function, approximated by linear interpolation with a 20-element table. This method gives a good approximation to the normal distribution in its tails, which can be important in preventing spurious convergence.

---

* Algorithm 10 could also have been applied to Tasks 1–3 by using $r[t] = 0$ for failure instead of $r[t] = -1$, but in this case Algorithm 10 would be equivalent to Algorithm 1.

According to (1), any value of $w[t]$ determines a legitimate value for $\pi[t]$, i.e., one between 0 and 1. Algorithms that update $w[t]$, such as Algorithms 4–9, are therefore relatively easily applied to a wide range of different tasks. Unlike the algorithms that update $\pi[t]$ directly, one need never worry about $\pi[t]$ exceeding the allowed range for a probability. Algorithms 4–9 are the only algorithms applied to all 6 tasks.

Algorithms 4–9 include both reinforcement-comparison algorithms (6–9) and non-reinforcement-comparison algorithms (4 and 5) of several different types. Discussion of their differences and similarities is deferred until the discussion of the results of the simulation experiments.

Each algorithm is parameterized by a single parameter α that determines the learning rate. For small positive values of α, learning is slow. For larger values of α, learning is faster, but if α is too large, then too much weight is put on early experience and convergence onto the incorrect action is possible. Accordingly, performance is typically an inverted-U shaped function of α.

Results

For each algorithm and each task to which it was applied, simulation experiments were performed with 8 different values for the learning-rate parameter α. These 8 values are the non-positive powers of 2, 2^0 through 2^{-7}, i.e., $\alpha = 1.$, .5, .25, .125, .0625, .03125, .015625, and .0078125. For each algorithm, task, and learning constant, 200 simulation runs were made, each of 200 time steps. The 200 runs differed from each other only in the choice of the initial seed for the pseudorandom number generator.

At the end of each run the final probability with which the algorithm selected

Action 1, $\pi[201]$, was determined. Since for all tasks Action 1 is the better of the two actions, this probability provides a measure of the performance of the algorithm. Figures 2–7 are plots of the average value of this final probability, averaged over 200 runs, for each algorithm, task, and α value. Each figure presents all the averages for a single task. For example, Figure 2 plots all the data from all runs made with Task 1.

In most cases performance of each algorithm was an inverted-U shaped function of α.* Since many of these algorithms are quite different and use the α parameter in different ways, it is not valid to compare the performance of different algorithms at the same α values. The approach taken here is to compare the performance of different algorithms on a task each ***with its own best*** α ***value***, that is, each with the α value that resulted in the algorithm's best average performance on the task. Using this performance measure, Figure 8 compares all algorithms on all tasks.

Consider the performance of Algorithms 6–9, shown in the lower graphs of Figures 2–8. On Tasks 4–6, Algorithms 6–9 performed virtually identically. On Tasks 1–3, Algorithm 9 performed slightly better Algorithms 6–8, but the general trend in performance as α varies is very similar, and the best performances of all these algorithms are not greatly different. Since these algorithms performed so similarly, and since Algorithm 9 performed the best of them on all tasks, only data from Algorithm 9 is plotted in the upper graphs of each figure for comparison with the other algorithms.

The performance plots in the upper graphs of Figures 2–8 (due to Algorithms 1–5 and 9–10) show much wider variation than those in the lower graphs. On all tasks except Task 6, some algorithms performed very well, while others performed

---

* The major exceptions to this generalization are Algorithms 6–9 on Tasks 4–6, in which performance continued to improve with α over the entire range of α's tested. Performance would probably have continued to improve slightly for these algorithms if higher values of α had been tested, but the performance levels are already so high that the question is moot.

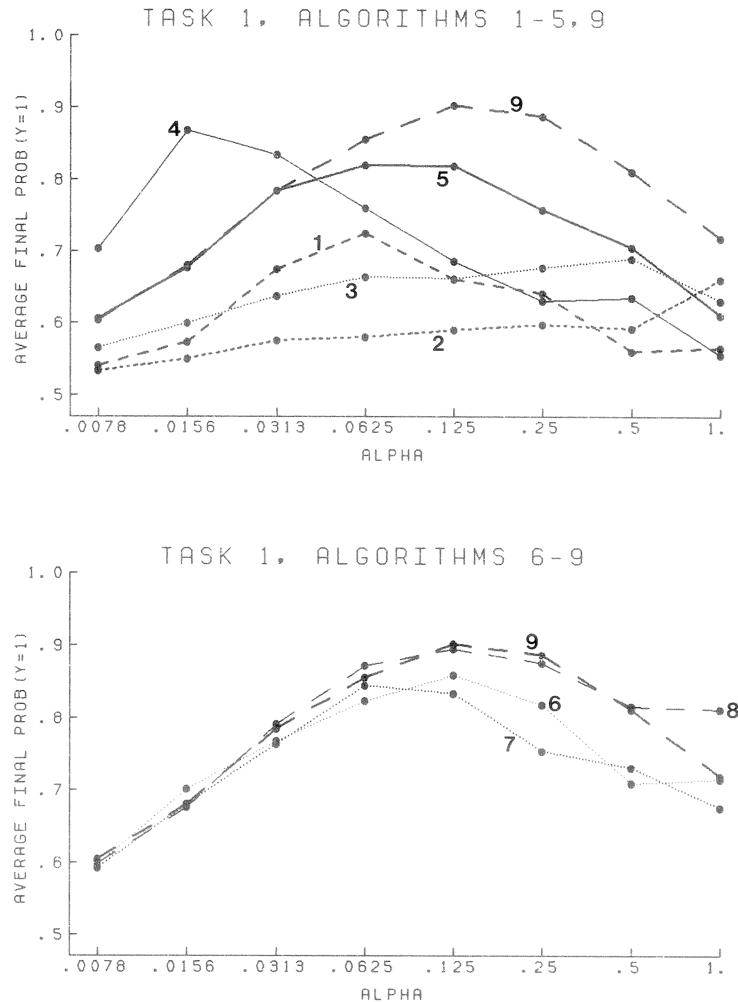


Figure 2. Algorithm Performance on Task 1 of Chapter II. Task 1 is a binary-reinforcement task with an imbalance of positive over negative reinforcement signal values. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the probability of selecting the correct action at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm.

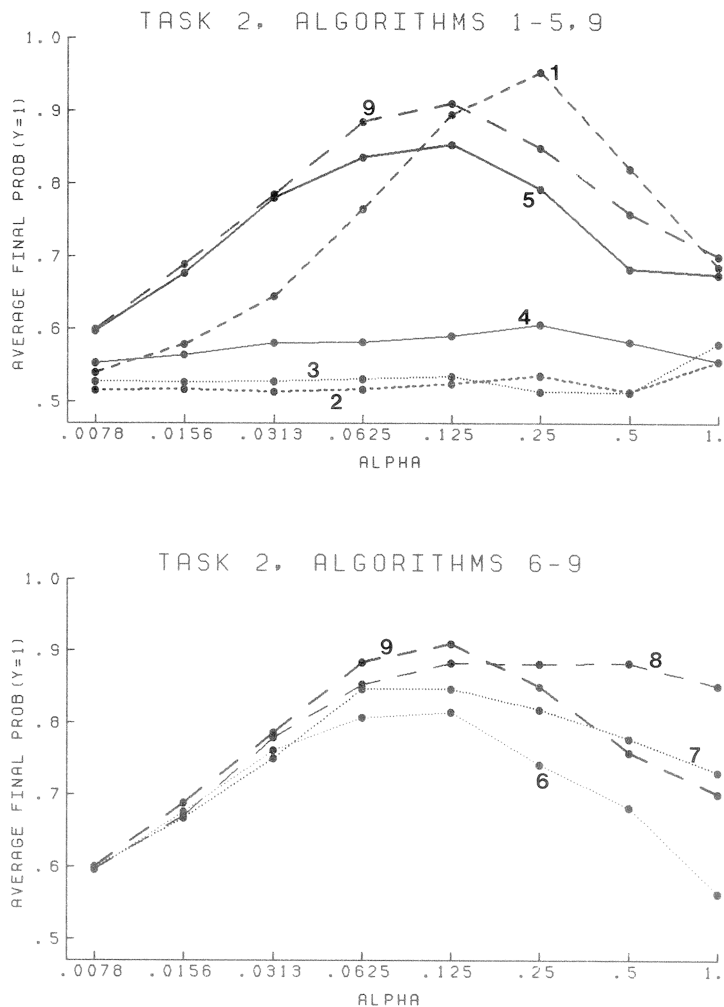
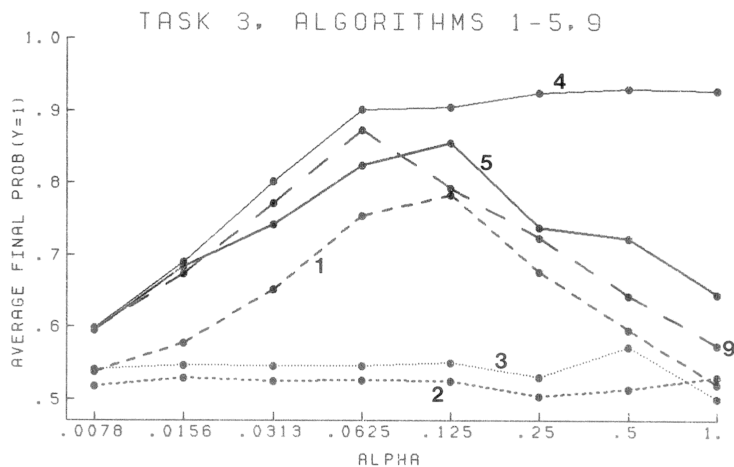


Figure 3. Algorithm Performance on Task 2 of Chapter II. Task 2 is a binary-reinforcement task with an imbalance of negative over positive reinforcement signal values. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the probability of selecting the correct action at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm.



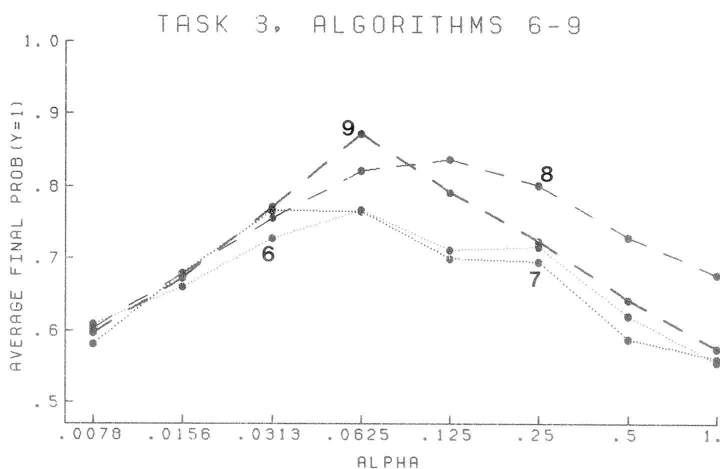
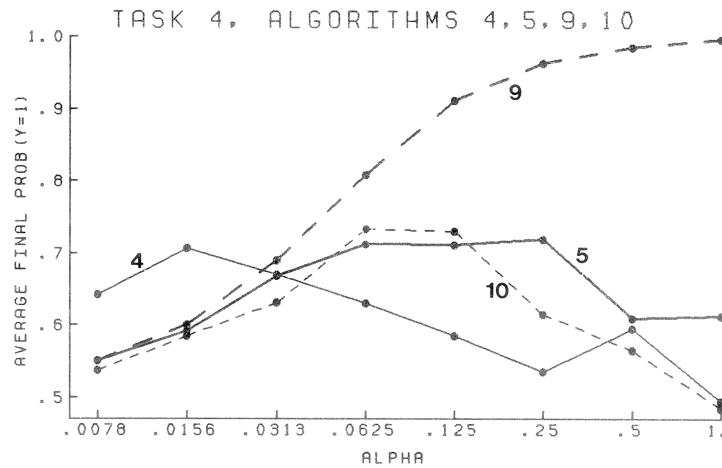


Figure 4. Algorithm Performance on Task 3 of Chapter II. Task 3 is a binary-reinforcement task with a balanced distribution of positive and negative reinforcement signal values. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the probability of selecting the correct action at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm.



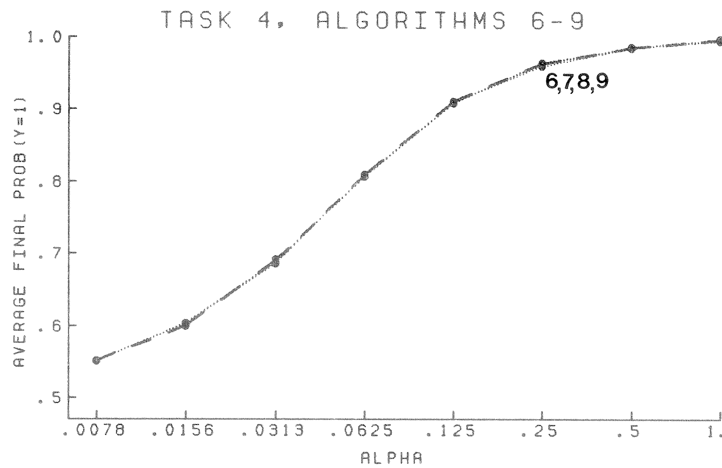
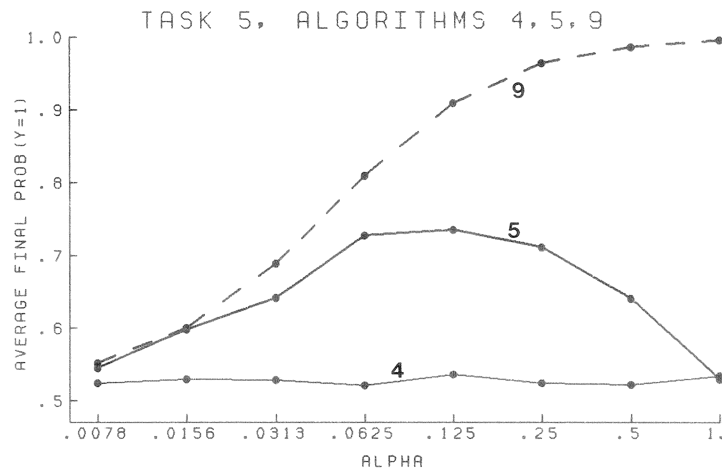


Figure 5. Algorithm Performance on Task 4 of Chapter II. Task 4 is a continuous-reinforcement task with an imbalance of positive over negative reinforcement signal values. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the probability of selecting the correct action at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm.



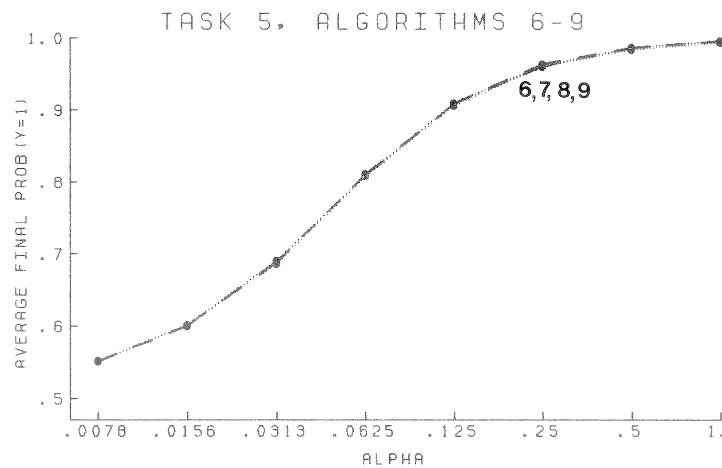
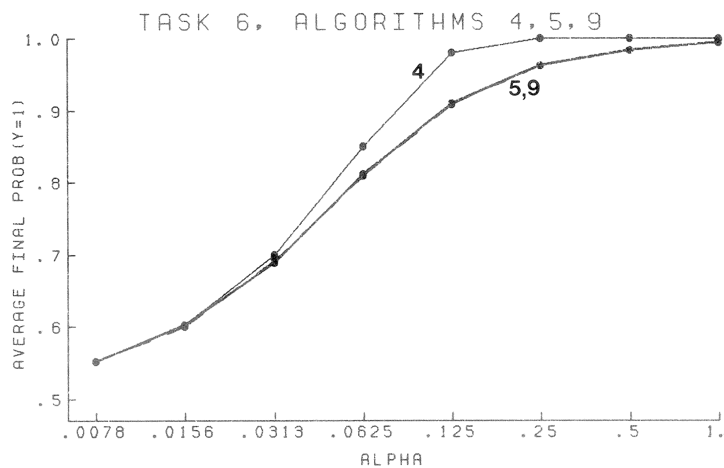


Figure 6. Algorithm Performance on Task 5 of Chapter II. Task 5 is a continuous-reinforcement task with an imbalance of negative over positive reinforcement signal values. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the probability of selecting the correct action at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm.



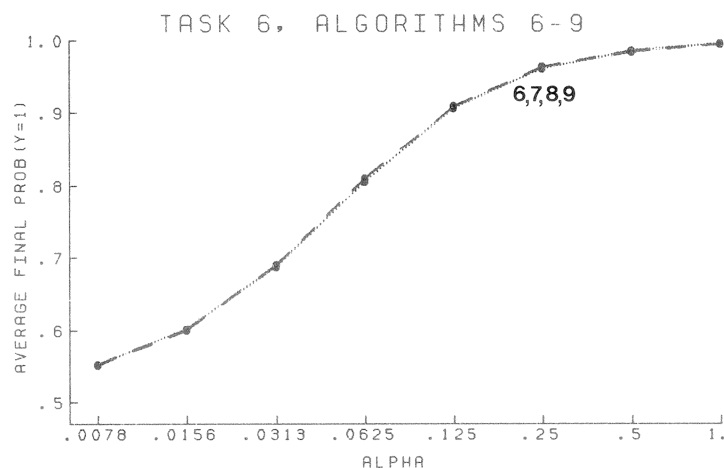


Figure 7. Algorithm Performance on Task 6 of Chapter II. Task 6 is a continuous-reinforcement task with a balanced distribution of positive and negative reinforcement signal values. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the probability of selecting the correct action at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm.

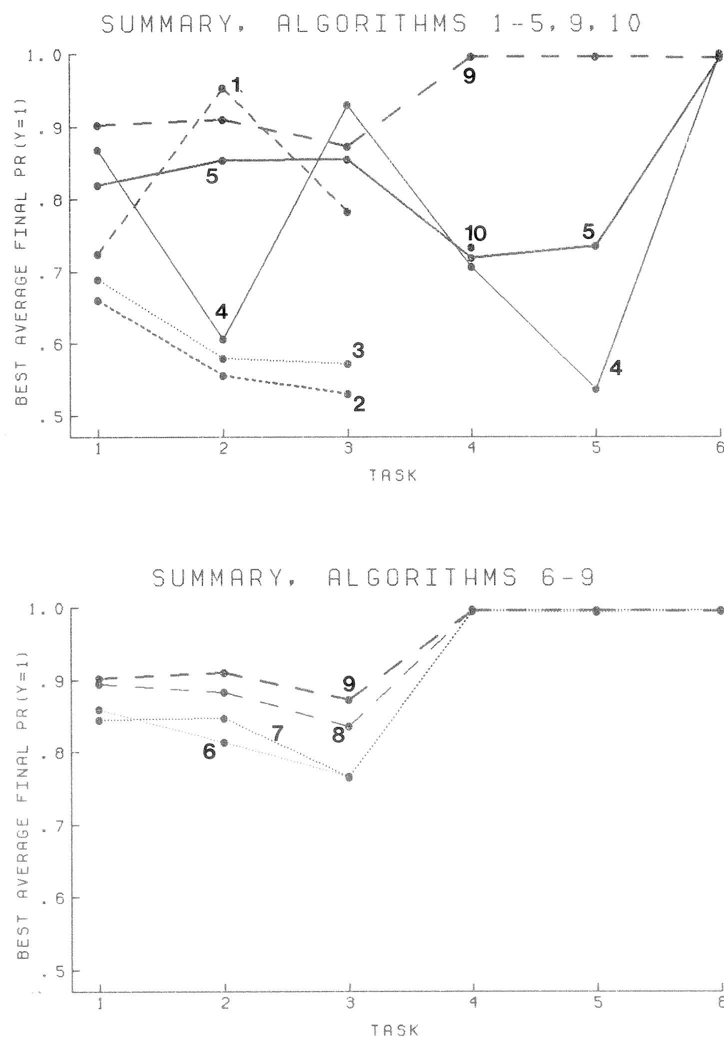


Figure 8. Summary of Algorithm Performance on all Tasks of Chapter II. Each point represents the performance level of a particular algorithm on a particular task with the α value at which performance was best for that algorithm on that task. Points due to the same algorithm (on different tasks) are connected by lines; the numeric label indicates the associated algorithm.

very poorly, varying from task to task. Algorithm 4, for example, performed the best of all algorithms on Tasks 3 and 6, but worst of all algorithms on Tasks 4 and 5, as well as performing very poorly on Task 2. Algorithm 1, on the other hand, performed the best of all algorithms on Task 2 and performed only "reasonably well" on Tasks 1 and 3, the only other tasks to which it was applied.

Of those algorithms whose performance is plotted in the upper graphs of Figures 2–8, the only one that performed well on all tasks is Algorithm 9. Algorithm 9 performed the best of all algorithms on Tasks 1, 4, and 5, tied for best on Task 6, and was a close second on the remaining Tasks 2 and 3.

To determine which differences between performances are statistically significant, a one-tailed t-test was used to determine the probability of obtaining each observed difference given the null hypothesis that underlying populations have the same mean value. According to this measure, almost all the performance differences shown in Figure 8 are highly statistically significant. For example, even the difference between the best performances of Algorithms 9 and 5 on Task 3, one of the smallest differences mentioned above, is statistically significant at the $P < .01$ level.

Although the one-tailed t-test is an appropriate statistic for these data, there are several reasons for caution in applying and interpreting it. First, there is sampling error in α: not all values of α were used in these experiments, and thus a statistically significant difference could appear between two algorithms merely because the optimal value was used for one while the optimal value for the other fell between those used. Fortunately, in these data there do not seem to be very large changes in performance from one value of α to the next value near the optimal α. Second, the t-test does not take into consideration the fact that only the best averages for each task and algorithm are compared. If the rest of the data had been included, more differences might be significant.

Discussion

Algorithms 6–9: Short-Term vs. Long-Term Reinforcement Comparison. Algorithms 6–9 are reinforcement-comparison algorithms. Algorithms 6 and 7 compare the current reinforcement and the preceding reinforcement, $r[t+1]-r[t]$. Algorithms 8 and 9 are somewhat more complex. Rather than compare current reinforcement with immediately preceding reinforcement, they compare it with a long-term measure of prior reinforcement. These algorithms compare current reinforcement with a real-valued variable called the ***expected level of reinforcement***, whose value at time t is denoted $p[t]$. That is, they update w according to $r[t+1]-p[t]$, where $p[t]$ is an exponentially-weighted average, or trace, of the preceding reinforcement values $r[t]$, $r[t-1]$, $r[t-2]$, ..., with greater weight going to more recent reinforcement values. In particular,

$$p[t+1] = p[t] + \beta(r[t+1] - p[t]) \quad \forall t \geq 0 \qquad \text{and} \qquad p[0] = 0, \tag{3}$$

where $0 < \beta \leq 1$. $\beta = .2$ for all simulation runs reported here. Since $p[t]$ is incremented fractionally according to its difference from $r[t+1]$, $p[t]$ tracks the level of $r[t+1]$. If r remains constant, p asymptotically approaches that constant level. The parameter β determines how quickly p matches a fixed level of r, and how quickly it tracks a changing level of r. A fairly low value for β, such as the .2 used in these experiments, implies that $p[t]$ tracks $r[t+1]$ relatively slowly. With $\beta = .2$, $p[t]$ is 20% (.2) due to $r[t]$, 16% ($.2 \cdot .8$) due to $r[t-1]$, 13% ($.2 \cdot .8^2$) due to $r[t-2]$, and 2% ($.2 \cdot .8^{10}$) due to $r[t-10]$, etc. If β is 1, then $p[t]$ exactly equals $r[t]$, and Algorithms 8 and 9 reduce identically to Algorithms 6 and 7.

The lower part of Figure 8 shows that the reinforcement-comparison algorithms with the long-term measure of preceding reinforcement (Algorithms 8 and 9) performed significantly better than those with the short-term measure (Algorithms 6

and 7) on Tasks 1–3, and identically on Tasks 4–6.

Algorithms 4–9: Structural Credit Assignment. For the moment consider Algorithms 4–9 in pairs, 4 with 5, 6 with 7, and 8 with 9. The algorithms of each pair are identical except for the factors of their update equations involving which action is taken. The even-numbered algorithms use $y[t] - \frac{1}{2}$ in this capacity whereas the odd-numbered algorithms use $y[t] - \pi[t]$. Both factors involve the action taken at time t, and the second factor also involves the action probability.

$y[t] - \frac{1}{2}$ is $\frac{1}{2}$ when Action 1 is chosen and $-\frac{1}{2}$ when Action 0 is chosen. The absolute value of this factor is constant, and its sign encodes which action was taken. The factor $y[t] - \pi[t]$ is somewhat more complex. On the earliest trials before $w[t]$ has changed very much from 0, $y[t] - \pi[t]$ operates exactly as $y[t] - \frac{1}{2}$ does because $\pi[t]$ is is near its initial value of $\frac{1}{2}$. As $w[t]$ moves significantly away from 0, $\pi[t]$ moves toward either 0 or 1, and $y[t] - \pi[t]$ begins to behave differently from $y[t] - \frac{1}{2}$. The two factors always have the same sign, but their absolute values differ. In comparison to $y[t] - \frac{1}{2}$, $y[t] - \pi[t]$ amplifies changes in $w[t]$ on those steps on which the less-likely action is taken, and diminishes changes in $w[t]$ on those steps on which the more-likely action is taken. For example, if $\pi[t]$ is .9, then $y[t] - \pi[t]$ is $1 - .9 = .1$ if the more-likely action, Action 1, is chosen, and $0 - .9 = -.9$ if the less-likely action, Action 0, is chosen.

It is not clear from the results shown in Figure 8 which algorithm of each pair is better. Algorithm 9 (using $y[t] - \pi[t]$) performed somewhat better than Algorithm 8 (using $y[t] - \frac{1}{2}$) on Tasks 1–3, and as well as Algorithm 8 on Tasks 4–6. Algorithm 7 (using $y[t] - \pi[t]$) in turn performed somewhat better than Algorithm 6 (using $y[t] - \frac{1}{2}$) on Task 2, worse on Task 1, and as well on Tasks 3–6. These differences are significant at the $P < .01$ level except for the differences on Task 1, which are significant only at the $P < .05$ level.

The differences between the performances of Algorithms 4 and 5 were much larger and more varied than those between the members of the other pairs of algorithms. Algorithms 4 and 5 performed nearly equally as well on Tasks 4 and 6 (the difference on Task 4 is not statistically significant; the difference on Task 6 is significant, but it probably would not have occurred if higher values of α had been tried for Algorithm 5). On the remaining tasks, Algorithm 4 (using $y[t] - \frac{1}{2}$) performed significantly better than Algorithm 5 (using $y[t] - \pi[t]$) on Tasks 1 and 3, and dramatically worse on Tasks 2 and 5.

What is the reason for the very poor performance of Algorithm 4 on Task 2? Task 2 is the binary-reinforcement task whose outcome is failure on almost all steps irrespective of the action chosen, although failure is slightly less likely (.8 versus .9) if Action 1 is chosen. Algorithm 4 uses the following update rule:

$$w[t+1] = w[t] + \alpha\, r[t+1](y[t] - \frac{1}{2}),$$

where $r[t+1] \in \{-1, 1\}$ on Task 2. This algorithm moves w in the right direction—positively—on Task 2, but as it starts to favor Action 1, the movement of w begins to slow down and ultimately stops. The problem is that as the algorithm starts to choose Action 1 more often than Action 0, most failures occur on Action 1 steps simply because Action 1 steps occur more frequently, and most steps end in failure in any case. The equilibrium probability of selecting Action 1 — that value of π at which the expected value of $w[t+1]$ given $w[t]$ is the same as $w[t]$ — for Algorithm 4 on Task 2 is readily computed. At equilibrium,

$$E\{w[t+1] \mid w[t]\} = w[t],$$

where $E\{\cdot \mid \cdot\}$ denotes a conditional expectation. In other words,

$$E\{\Delta w[t] \mid w[t]\} = E\{w[t+1] - w[t] \mid w[t]\} = 0.$$

Using $P\{\cdot \mid \cdot\}$ to denote a conditional probability,

$$\begin{aligned}
E\{\Delta w[t] \mid w[t]\} &= P\{y[t]=1 \mid w[t]\}P\{r[t+1]=1 \mid y[t]=1\}\,\alpha\,1(1-.5) \\
&\quad + P\{y[t]=1 \mid w[t]\}P\{r[t+1]=-1 \mid y[t]=1\}\,\alpha\,(-1)(1-.5) \\
&\quad + P\{y[t]=0 \mid w[t]\}P\{r[t+1]=1 \mid y[t]=0\}\,\alpha\,1(0-.5) \\
&\quad + P\{y[t]=0 \mid w[t]\}P\{r[t+1]=-1 \mid y[t]=0\}\,\alpha\,(-1)(0-.5) \\
&= \pi[t].2\,\alpha\,.5 \\
&\quad + \pi[t].8\,\alpha\,(-.5) \\
&\quad + (1-\pi[t]).1\,\alpha\,(-.5) \\
&\quad + (1-\pi[t]).9\,\alpha\,.5 \\
&= .5\alpha\Big[.1+.9+\pi[t]\,(.2-.8+.1-.9)\Big] = 0,
\end{aligned}$$

implying that

$$\pi[t] = 4/7 \simeq .57.$$

Figure 3 shows that the final values of π for Algorithm 4 were in fact all near .57.

Algorithm 5 differs from Algorithm 4 only in the replacement of the $y[t] - \frac{1}{2}$ factor by $y[t] - \pi[t]$, and yet this is enough to completely solve the problem described above. As Algorithm 5 starts to choose Action 1 more often, it too starts experiencing more steps on which it chooses Action 1 and whose outcome is failure, but because of the $y[t] - \pi[t]$ factor, the effect of each of these failures is diminished. Unlike Algorithm 4, Algorithm 5 results in the probability of choosing the correct action continuing to increase towards 1 on Task 2.

In conclusion, these results show that algorithms using $y[t] - \pi[t]$ sometimes show improved and sometimes degraded performance over the corresponding algorithms using $y[t] - \frac{1}{2}$. Where $y[t] - \pi[t]$ improves performance, it sometimes improves it dramatically, and where it degrades performance, it degrades it relatively little. Finally, for the best performing algorithms (8 and 9), the $y[t] - \pi[t]$ factor results in a small consistent improvement across tasks.

Algorithms 1–10: Reinforcement-Comparison Mechanisms. The algorithms with reinforcement-comparison mechanisms (Algorithms 6–9) performed much better across tasks than the comparable algorithms without reinforcement-comparison mechanisms (Algorithms 4 and 5). Although Algorithm 4 performed best of all algorithms by small margins on the balanced-reinforcement tasks (3 and 6), it performed far worse than the reinforcement-comparison algorithms on Tasks 2, 4, and 5. On Tasks 4 and 5 all reinforcement-comparison algorithms showed complete and correct learning, and all other applicable algorithms showed only a very low level of learning. These results indicate that reinforcement-comparison mechanisms improve performance on unbalanced-reinforcement tasks (1, 2, 4, and 5), and at least do not degrade performance on balanced-reinforcement tasks (3 and 6).

Algorithms 1, 2, 3, and 10 are learning automata algorithms commonly discussed in the literature. Although these algorithms are not particularly noted for their speed of learning, they provide useful reference points with which to compare the performance of the other algorithms. Algorithms 1–3 could be applied only to the binary-reinforcement tasks (1–3). Algorithms 2 and 3 were the poorest performers on these tasks, and Algorithm 1, on balance, performed significantly worse than several of the other algorithms, including particularly the best reinforcement-comparison algorithm, Algorithm 9. Algorithm 10 was the only learning-automaton algorithm applied to Task 4. However, it did no better than the other algorithms in challenging the greatly superior performance of the reinforcement-comparison algorithms on the continuous-reinforcement tasks.

Absolute Expediency of a Reinforcement-Comparison Algorithm

This section presents a proof regarding the asymptotic convergence of a reinforcement-comparison algorithm. Such convergence results are often difficult to obtain for algorithms, such as reinforcement-comparison algorithms, which do not adopt the independence-of-path assumption. The following proof illustrates that such results are at least possible in special cases.

Consider the reinforcement-comparison algorithm defined by:

$$\pi[t+1] = \pi[t] + \Delta\pi[t] \qquad \text{where} \qquad \Delta\pi[t] = \alpha(r[t+1] - r[t])(y[t] - \pi[t]), \quad (4)$$

$0 < \alpha < 0.5$, $\pi[t]$ is the probability that $y[t] = 1$, and $r[t+1] \in \{-1, 1\}$ is success or failure depending on the action $y[t] \in \{0, 1\}$. Performance $J[t]$ at time t is defined as the expected value of the reinforcement signal at time t:

$$J[t] = E\{r[t+1] \mid \pi[t]\}.$$

An algorithm for generating successive values for π is normally said to be ***absolutely expedient*** if and only if performance increases in expected value from step to step:

$$E\{J[t+1] \mid \pi[t]\} > J[t], \quad (5)$$

for all t, for all values of $\pi[t]$, and for all possible binary-reinforcement tasks (i.e., for all (non-equal) success probabilities for the two actions) (cf. e.g., Narendra and Thathachar, 1974).

The criterion of absolute expediency given by (5) is not well-defined for the algorithm given by (4), because, in using the expression $E\{J[t+1] \mid \pi[t]\}$, (5) implicitly assumes that the expected value of $J[t+1]$ is well-defined given only $\pi[t]$. For algorithms such as (4), which include memory variables other than $\pi[t]$, this is not so. Although $E\{J[t+1] \mid \pi[t]\}$ is not defined for this algorithm,

$E\{J[t+1] \mid \pi[t], \pi[t-1]\}$ is. It is this quantity which is shown here to increase from step to step. That is, henceforth a criterion is used which considers an algorithm to be absolutely expedient if and only if:

$$E\{J[t+1] \mid \pi[t], \pi[t-1]\} > J[t], \tag{6}$$

for all t, for all values of $\pi[t]$ and $\pi[t-1]$, and for all binary-reinforcement tasks.

Theorem. *The algorithm given by (4) meets the criterion of absolute expediency given by (6).*

Let

$$p_1 = P\{r[t+1] = 1 \mid y[t] = 1\} \qquad \text{and} \qquad p_0 = P\{r[t+1] = 1 \mid y[t] = 0\}.$$

Then

$$\begin{aligned} J[t] &= E\{r[t+1] \mid \pi[t]\} \\ &= p_1\pi[t] + p_0(1-\pi[t]) - (1-p_1)\pi[t] - (1-p_0)(1-\pi[t]) \\ &= 2p_0 - 1 + 2\pi[t](p_1 - p_0). \end{aligned}$$

Since $J[t+1]$ is a linear function of $\pi[t+1]$, it suffices to show that $E\{\Delta\pi[t] \mid \pi[t], \pi[t-1]\}$ is of the right sign, i.e., is positive if $p_1 > p_0$, and negative if $p_1 < p_0$. To compute this expected value one must consider 8 cases, one for each of the possible combinations of values for $y[t]$, $r[t+1]$ and $r[t]$:

$$
\begin{aligned}
&E\{\Delta\pi[t] \mid \pi[t], \pi[t-1]\} = \\
&\qquad \pi[t]\, p_1\, P\{r[t] = 1 \mid \pi[t-1]\}\, \alpha\, 0(1-\pi[t]) \\
&\qquad + \pi[t]\, p_1\, P\{r[t] = -1 \mid \pi[t-1]\}\, \alpha\, 2(1-\pi[t]) \\
&\qquad + \pi[t]\, (1-p_1)\, P\{r[t] = 1 \mid \pi[t-1]\}\, \alpha\, (-2)(1-\pi[t]) \\
&\qquad + \pi[t]\, (1-p_1)\, P\{r[t] = -1 \mid \pi[t-1]\}\, \alpha\, 0(1-\pi[t]) \\
&\qquad + (1-\pi[t])\, p_0\, P\{r[t] = 1 \mid \pi[t-1]\}\, \alpha\, 0(0-\pi[t]) \\
&\qquad + (1-\pi[t])\, p_0\, P\{r[t] = -1 \mid \pi[t-1]\}\, \alpha\, 2(0-\pi[t]) \\
&\qquad + (1-\pi[t])\, (1-p_0)\, P\{r[t] = 1 \mid \pi[t-1]\}\, \alpha\, (-2)(0-\pi[t]) \\
&\qquad + (1-\pi[t])\, (1-p_0)\, P\{r[t] = -1 \mid \pi[t-1]\}\, \alpha\, 0(0-\pi[t]) \\
&\Big(\text{Letting } q = P\{r[t] = 1 \mid \pi[t-1]\}\Big)
\end{aligned}
$$

$$
\begin{aligned}
&= \pi[t](1-\pi[t])\, \alpha\, 2\Big(p_1(1-q) - (1-p_1)q - p_0(1-q) + (1-p_0)q\Big) \\
&= \alpha\, 2\pi[t](1-\pi[t])(p_1 - p_0).
\end{aligned}
$$

Since $\pi[t]$ is always between 0 and 1, this expression is always of the same sign as $p_1 - p_0$. Q.E.D.

Extensions

The simulation experiments reported here need to be extended and verified in several ways. A great many more algorithms should be tried, including other algorithms from the learning automata literature, algorithms from the n-armed bandit literature, and reinforcement-comparison algorithms other than those tried here. For example, an algorithm that forms separate estimates of the expected reinforcement for each action, and then compares them to determine its action

probability is a reinforcement-comparison algorithm, though different from those compared here, and would make an interesting addition to this study.

The only real difficulty in extending the experimental work in the direction of additional algorithms is the great number of possible algorithms and the large number of experiments that would need to be run. In the experiments reported here only a single parameter of each algorithm was varied to determine its highest performance level, but for other algorithms 2 or even 3 parameters may need to be varied.

The experiments reported here also need to be extended to other tasks. Other variants of the binary-action with binary or continuous-reinforcement tasks investigated here would be useful, but it is probably more important to experiment with other classes of tasks. Tasks with more than 2 actions or more than 2 discrete reinforcement levels (called Q-model environments in the LAT literature) come to mind.

Finally, further mathematical analysis is also clearly called for. The superior performance of reinforcement-comparison algorithms in these simulations suggests that such algorithms may be worth analyzing, despite the difficulties. Only a single minor result has been presented here; other mathematical results relating to the convergence of reinforcement-comparison algorithms, or to the learning-rate of such algorithms, would be of great interest.

Conclusions

These results strongly suggest that learning automata theorists and mathematical learning theorists may have excluded some of the fastest learning algorithms by not considering reinforcement-comparison algorithms. On unbalanced-

reinforcement tasks, reinforcement-comparison algorithms tend to learn much more rapidly than analogous algorithms without reinforcement-comparison mechanisms.

These results do not prove that reinforcement-comparison algorithms are superior to non-reinforcement-comparison algorithms. Strictly speaking, the conclusion that reinforcement-comparison mechanisms greatly improve learning rate applies only to this particular set of algorithms and tasks. However, these results cannot help but suggest that remembering past reinforcement levels for subsequent comparison may be an essential feature of robust and high-performance learning algorithms for discrete-action nonassociative reinforcement-learning tasks.

CHAPTER III

ASSOCIATIVE LEARNING

Associative-learning tasks are tasks in which a mapping from input to output must be formed by the learning system. Stimuli and actions must be associated in such a way that the occurrence of a stimulus evokes the most suitable or appropriate action. Learning in pattern classification, for example, is associative learning because associations are learned between patterns (stimuli) and classifications (actions). However, pattern-classification learning is typically not reinforcement learning because the learning system is not required to search for the best classifications — these are provided by a "teacher" during a training sequence. In nonassociative-learning tasks, such as those considered in the preceding chapter, the learning system need only *find* the best action, it need not associate it with any stimulus.

The word "associative" in associative learning is not meant to suggest that it is necessarily symmetrical, non-symbolic, or distributed, but only that what is learned is an association or linking of actions to stimuli. Any such other connotations of the word "associative" are coincidental. This kind of learning could equally well be called "mapping learning," to emphasize that a set of relationships, or "map," is learned, or it could equally well be called "learning with stimuli" to emphasize that this type of learning always involves non-reinforcing stimuli used to *select* actions as well as reinforcement input to *evaluate* them.

The experiments described in this chapter compare the performance of 11 associative-learning algorithms on 12 associative reinforcement-learning tasks. The algorithms are generalizations of those studied in Chapter II. The experiments focus on problems of misleading generalization occurring when similar stimuli must be discriminated. In Chapter II, reinforcement-comparison mechanisms were shown to improve performance on unbalanced-reinforcement tasks. One purpose of the experiments of Chapter III is to determine whether they ease the problem of misleading generalizations as well.

Approaches to Associative Learning

There have been at least three approaches to the problem of associating actions with stimuli: The *independent-associations* approach, the *stimulus-sampling-theory* approach, and the *linear-mapping* approach. Each of these approaches to associative learning and the relationships between them are discussed below. A fourth approach, not discussed here, is based on matching stimuli to *prototypes*, and selecting actions based on which prototype matches best (e.g., Reilly, Cooper and Elbaum, 1982; Hampson and Kibler, 1982; Spinelli, 1970; Anderson et.al., 1977).

An important issue in evaluating approaches to associative learning is that of stimulus similarity. How stimulus similarity is handled determines when generalization between stimuli occurs and when discrimination is possible. *Generalization* between stimuli occurs when training to one stimulus transfers and influences behavior (action selections) to another stimulus. When the transfer is positive, i.e., when the action learned in the presence of the first stimulus becomes more likely to occur in the presence of the second stimulus as a result of the training, then the two stimuli are said to be similar. Stimulus *discrimination* refers to the ability of a system to learn to perform two different actions in response to two different

stimuli. In stimulus discrimination, one is usually concerned with discriminating between stimuli that are similar, in which case generalization between them may make their discrimination difficult. The ability to ***perfectly discriminate*** stimuli implies the ability to overcome any generalization between them and to behave differently to them with 100% reliability.

The Independent-Associations Approach. One approach to associative learning is to make a separate association to action for every stimulus presented to the learning system. Since each association is distinct and separate from every other, a separate memory variable (or variables) is required for each stimulus. To the extent memory is limited, this requirement restricts the usefulness of this approach to tasks with comparatively small numbers of distinguishable stimuli. The independent-associations approach has been used in reinforcement learning control theory (Mendel and McClaren, 1970), and in mathematical learning theory (Bush and Mosteller, 1955), and occasionally in learning automata theory (Lakshmivarahan, 1981; Witten, 1977) and artificial intelligence (Michie and Chambers, 1968a,b). It is also the basis of "table-lookup" approaches to storing mappings (Raibert, 1978). Because all stimuli are treated separately, no stimuli are similar and no generalization can occur, and hence stimulus discrimination is trivial.

By ignoring the issues of stimulus similarity, generalization, and discrimination, the independent-associations approach gives up the opportunities they provide. Transfer of training due to generalization can greatly increase the rate of learning (assuming similar stimuli should elicit similar actions). In complex learning tasks it may be rare for the exact same stimulus to occur twice in the lifetime of the learning system. In such cases, generalization between stimuli is essential.

On the other hand, an advantage of the independent-associations approach is that it permits one to apply reinforcement-learning algorithms known to work on

nonassociative tasks directly to associative tasks. To do this, one allots a separate instance of the nonassociative algorithm for each stimulus that will ever be presented to the learning system. Whenever a stimulus occurs, the corresponding algorithm takes one step. This reduces the associative task to a set of independent nonassociative tasks. Using the independent-associations approach and this technique, any algorithm known to work for nonassociative tasks can be applied with confidence to associative tasks as well. For this reason the independent-associations approach has remained of considerable theoretical interest despite its requirement for small numbers of distinct stimuli and its lack of generalization capabilities.

A slight modification of the independent-associations approach as discussed thus far is to make independent associations not with stimuli, but with groups of stimuli, where the grouping of stimuli is determined before training. This modification replaces the requirement of a small number of different stimuli with the requirement of a small number of groups of stimuli. For example, Michie and Chambers's "Boxes" system (Michie and Chambers, 1968a) uses this approach to divide a continuous state space constituting an infinite number of stimuli into 162 groups. This sort of modification introduces a primitive form of stimulus similarity and generalization. Stimuli within a group are all treated as the same stimulus, so complete generalization occurs between them, while discrimination between them is impossible. The biggest problem with this approach is deciding how the stimuli are to be grouped. Which stimuli should be grouped together and which need to be treated separately is usually an important part of the problem.

The Stimulus-Sampling-Theory Approach. One approach to associative learning which does include stimulus similarity and generalization is that of stimulus sampling theory (Estes, 1950, 1959a,b). In stimulus sampling theory (SST), stimuli correspond to subsets of a large number of independently variable stimulus components. (In SST these are usually called stimulus elements, but I will refer

to them here as stimulus components.) Each stimulus component is associated in an all-or-none fashion with a particular action. The proportion of stimulus components in a stimuli's subset associated with an action determines the probability of that action's being chosen in response to the stimulus. When a stimulus is presented, a random sample is drawn from the corresponding subset of stimulus components. As a consequence of a conditioning (learning) rule, the stimulus components sampled may have their associations with action changed.

In SST two stimuli are said to be similar to the extent that their subsets have common stimulus components. Generalization in SST works as follows. Training with one stimulus increases the proportion of the stimulus components in its subset associated with a particular action. If a second similar stimulus is then presented, the probability of its eliciting the action is increased because a higher proportion of the stimulus components it has in common with the first stimulus are conditioned to the action. If all subsets are disjoint, then no generalization can occur. In this special case, the SST approach reduces to the independent-associations approach.

Although SST's approach to similarity works out fairly well for generalization (La Berge, 1961; Carterette, 1961; Atkinson and Estes, 1963), it has greater difficulty with discrimination learning. Under the SST approach a learning system can never perform different actions to two similar stimuli with 100% reliability. The common stimulus components of the similar stimuli cannot be allocated all to both actions. Thus some proportion of at least one of the stimuli's components must be associated with the wrong action for that stimulus. Since in SST this proportion directly determines the probability of selecting each action, perfect discrimination is impossible.

Several directions have been taken in SST to handle this problem with discrimination learning. One is based on the idea of somehow eliminating the effectiveness of stimulus components in the intersection of stimulus subsets when discrimina-

tion between the corresponding stimuli becomes necessary. Work along this line is usually based on the idea of selective attention (Restle, 1955; Zeaman and House, 1963; Lovejoy, 1968; Sutherland and Mackintosh, 1971). The other major direction taken to handle stimulus discrimination within SST is based on the idea that configurations of stimulus components may be perceived as whole patterns or "gestalts" (Atkinson and Estes, 1963; Friedman, Trabasso and Mosberg, 1967). Both of these remedies involve bringing in additional mechanisms to handle the problem of discrimination learning.

The Linear-Mapping Approach. A third approach to stimulus similarity has been widely used in pattern classification, control theory, neural-network and associative-memory research, and animal learning theory. In the linear-mapping approach, stimuli are represented as vectors, where each component of the vector indicates the presence or absence (or extent) of a stimulus component. A linear mapping is formed from the stimulus vectors to a quantity that determines the action to be taken. As a consequence, similarity and generalization between stimulus vectors is determined by the degree of their linear dependence, i.e., by their inner product.

If the set of stimulus vectors is orthogonal, no stimuli are similar, and no generalization occurs. In this case, the linear-mapping approach reduces to a rather roundabout form of the independent-associations approach.

The linear-mapping approach is the approach used in the research reported here. The experiments described in this chapter are concerned exclusively with binary-action tasks in which the action $y[t]$ is either 1 or 0. In the binary-action case of the linear-mapping approach, actions are selected as follows:

$$y[t] = \begin{cases} 1, & \text{if } s[t] + \eta[t] > 0; \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

where

$$s[t] = \sum_{i=1}^{n} x_i[t] w_i[t], \tag{8}$$

$\eta[t]$ is a normally distributed random variable of mean 0 and standard deviation σ_y, n is the total number of stimulus components, $\vec{x}[t] = (x_1[t], x_2[t], \ldots, x_n[t])$ is the stimulus vector presented at time t, and $\vec{w}[t] = (w_1[t], w_2[t], \ldots, w_n[t])$ is a modifiable weight vector, called the ***action-association vector***. For the special case in which the same stimulus vector is presented on every time step, and in which that vector has only one component, whose value is 1, (7) and (8) reduce exactly to (2), the nonassociative action-selection equation used in Chapter II.

For cases with more than two possible actions, there are two subapproaches within the linear-mapping approach. The ***distinct-actions subapproach*** requires that there be a finite number of distinct actions, and the ***component-actions subapproach*** breaks actions up into independently-variable components, each of which may be either binary or continuous. These two subapproaches are briefly described below, but are not considered elsewhere in this dissertation.

The Distinct-Actions Subapproach. In the distinct-actions subapproach, each stimulus component has a degree of association with each action. Let $w_{ij}[t]$ denote the strength of the association between stimulus component i and action j at time t, and let $W[t] = \left[w_{ij}[t]\right]$ denote the $n \times m$ matrix of these associations where there are n stimulus components and m possible actions. The action $y[t]$ is selected based on the product of $W[t]$ and the current stimulus vector, denoted $\vec{x}[t] = (x_1[t], x_2[t], ..., x_n[t])$. This product is an m-vector denoted $\vec{s}[t] = (s_1[t], s_2[t], ..., s_m[t])$:

$$\vec{s}[t] = W[t]\vec{x}[t],$$

or, equivalently,

$$s_j[t] = \sum_{i=1}^{n} x_i[t] w_{ij}[t], \qquad \forall j. \tag{9}$$

The simplest technique for selecting the action $y[t]$ from $\vec{s}[t]$ is to choose the action corresponding to the largest component of $\vec{s}[t]$:

$$y[t] = j, \qquad \text{where} \qquad s_j[t] > s_k[t], \quad \forall k \neq j.$$

In reinforcement learning, however, it is usually better to choose probabilistically rather than deterministically. A simple probabilistic rule is to add an independent random variable $\eta_j[t]$ to each component of $\vec{s}[t]$ and then select the action corresponding to the component whose $s_i[t] + \eta_j[t]$ is largest:

$$y[t] = j, \qquad \text{where} \qquad s_j[t] + \eta_j[t] > s_k[t] + \eta_k[t], \quad \forall k \neq j,$$

and each $\eta_i[t]$ is chosen from a normal distributtion with mean zero and standard deviation σ_y.

The Component-Actions Subapproach. A second linear-mapping subapproach involves representing the actions as consisting of independently-variable action components in the same way that stimuli are represented as consisting of stimulus components. In this subapproach, actions are vectors of action components, and each stimulus component has a degree of association, either positive or negative, with each action component. Here $w_{ij}[t]$ denotes the strength of the association between stimulus component i and action component j at time t, and $W[t] = \left[w_{ij}[t]\right]$ denotes the $n \times m$ matrix of these associations where there are n stimulus components and m action components. $\vec{s}[t]$ is computed as in (9). The simplest rule for determining the action vector $\vec{y}[t] = (y_1[t], y_2[t], ..., y_m[t])$ is

$$y_j[t] = \begin{cases} 1, & \text{if } s_j[t] + \eta_j[t] > 0; \\ 0, & \text{else,} \end{cases} \qquad \forall j.$$

If a continuous-valued action vector ($\vec{y}[t] \in \Re^m$) is desired, then $\vec{y}[t]$ is determined by

$$y_j[t] = s_j[t] + \eta_j[t],$$

where each $\eta_j[t]$ is an independent normally-distributed random variable with mean 0 and standard deviation σ_y.

Discrimination in the Linear-Mapping Approach. The linear-mapping approach is better at discrimination than is the SST approach. In the SST approach, perfect discrimination between two stimuli is possible only if the stimuli are totally dissimilar. In the linear-mapping approach, two vectors can be very similar, and perfect discrimination will still usually be possible. Roughly, two stimulus vectors are similar in the linear-mapping approach according to the angle between them. As long as the angle between two vectors is not zero, it is possible to fit a hyperplane between them and thereby discriminate between them. Therefore, very similar vectors can be mapped to completely different actions with as high a degree of reliability as desired.

Although the linear-mapping approach is in some respects a better approach to associative learning than either SST or the independent-associations approach, it obviously has limitations. For example, considered by itself the linear-mapping approach is not capable of forming nonlinear associations, or of arbitrarily categorizing linearly-dependent stimulus vectors. To do these things it must rely on some as yet unspecified additional mechanisms, as must SST in order to achieve perfect discrimination.

Nevertheless, this study is concerned only with the linear-mapping approach, despite these limitations, because even the linear problem remains largely unsolved for associative reinforcement learning. The experiments presented in this chapter suggest that some associative reinforcement-learning algorithms that have previously been investigated (e.g., Farley and Clark, 1954) fail when required to form linear maps in interaction with certain types of environments. The forming of linear maps for non-reinforcement associative learning has been well worked out

(see e.g., Rosenblatt, 1957, 1962; Widrow and Hoff, 1960; Nilsson, 1965; Duda and Hart, 1973), and researchers in this area have gone on to consider nonlinear mapping (e.g., Poggio, 1975; Reilly, Cooper and Elbaum, 1982). The strategy of first studying linear mapping has proven to be a successful one for investigating associative learning that is not reinforcement learning; it is probably a good strategy for associative reinforcement learning as well.*

The linearity or nonlinearity of a particular mapping is dependent on the way stimuli are represented as stimulus vectors. What is nonlinear in one representation may be linear in another and vice-versa. For example, in the extreme case, any mapping can be made linear by representing stimuli as vectors only one component of which is nonzero, a different component for each stimulus, i.e., by reducing the linear-mapping approach to an independent-associations approach as discussed earlier. In this case, since any stimulus can be mapped to any action by changing the association of the stimuli's dedicated component, any mapping can be formed. Although this extreme approach, as mentioned earlier, is rarely useful, it illustrates how strongly the linearity or nonlinearity of a mapping depends on the stimulus representation. Any mapping can be implemented as a linear mapping with an appropriately chosen representation.

When a mapping that is nonlinear in the original representation is required of a system employing a linear mapping approach, some other mechanism or mechanisms must be relied on to change the representation so that the mapping becomes linear. The problems in doing this are closely related to the knowledge-representation problem ephasized in AI (see e.g., Barr and Feigenbaum, 1981). It is well known in AI that the way knowledge is represented is crucial in determining the effectiveness of an intelligent system. This is just as true for learning systems.

---

* For one approach to nonlinear mapping in associative reinforcement learning, and further discussion of this problem, see (Barto, Anderson, and Sutton, 1982; Anderson, 1982; Barto, in press).

Among other things, a change in representation can completely change which stimuli are similar, dissimilar, or unrelated, and thereby change an easy learning task into a hard one or vice-versa. A complex learning system, even if it relies on a linear-mapping approach to form associations, must be able to find good representations if it is to be maximally effective in the long run and over a wide range of tasks. That mappings can be changed from nonlinear to linear by a change in representation, and that such changes in representation must occur in any event, provides additional justification for studying purely linear mappings.

How good representations might be found and modified is a large and complex subject which is beyond the scope of this work. In terms of the division of the credit-assignment problem, this issue falls in the area of structural rather than temporal credit assignment, because it involves building and assigning credit to internal knowledge-representation structures.

Finally, it is important to emphasize that the three approaches to associative learning discussed above are ways of associating actions with stimuli, not algorithms for deciding which associations to make. The differences among approaches discussed above are largely independent of what learning algorithm is used. If an approach is lacking in some capability, no learning algorithm can save it from the deficiency. Conversely, if an approach has superior capabilities, then a learning algorithm must be found to implement those capabilities before the superiority can be considered significant.

Independent-Step Associative Learning

The experiments described in this chapter involve *independent-step* associative-

learning tasks. An independent-step task is one in which the interaction between the learning system and its environment can be separated into steps, each of which is independent of all others. In associative reinforcement learning, each step involves the presentation of one stimulus vector to the learning system, its selection of one action, and finally the delivery of a reinforcement feedback signal by the environment. Such steps are said to be independent if the stimulus, action, and reinforcement of one step do not influence the stimulus or reinforcement of any other step. In particular, the action selected on one step may influence the reinforcement on that step but not that of any other step.

Formally, the restriction to independent-step learning tasks means that the environment does not change state. For these tasks the environment is characterized by a sequence of stimuli, $x[1]$, $x[2]$, ..., and a map ρ such that

$$\rho: X \times Y \rightarrow \Re$$

$$\rho(x[t], y[t]) \mapsto r[t+1],$$

where X is the stimulus space, Y is the action space, and the reinforcement space $\Re$ is the set of real numbers. As usual, ρ may be stochastic (probabilistic). A single step involves the stimulus at time t, $x[t]$, the action at time t, $y[t]$, and the reinforcement at time $t+1$, $r[t+1]$. Note that the reinforcement is denoted as occurring one time step later than the stimulus and the action. This convention is used because it is consistent with later descriptions of learning tasks that cannot be divided into independent steps.

Tasks

This section provides a complete specification of the 12 tasks used in the experiments of this chapter. Discussion of the rationale for using these particular tasks is deferred until the "Discussion" section. Table 3 summarizes the 12 tasks;

Table 3. Associative Learning Tasks.

Task	Expected Value of Reinforcement Stimulus $\vec{x}^1$ $y=1$	Stimulus $\vec{x}^1$ $y=0$	Stimulus $\vec{x}^2$ $y=1$	Stimulus $\vec{x}^2$ $y=0$	Stimulus Frequency $\vec{x}^1$	Stimulus Frequency $\vec{x}^2$	Stimulus Intensity $\vec{x}^1$	Stimulus Intensity $\vec{x}^2$	Steps	Description
1	.1(.55)	−.1(.45)	−.1(.45)	.1(.55)	.5	.5	1	1	1500	Symmetrical
2	.1	−.1	−.1	.1	.5	.5	1	1	200	
3	−.6(.2)	−.8(.1)	.6(.8)	.8(.9)	.5	.5	1	1	1500	Reinforcement Level
4	−.6	−.8	.6	.8	.5	.5	1	1	200	Asymmetry
5	.05(.525)	−.05(.475)	−.15(.425)	.15(.575)	.5	.5	1	1	1500	Reinforcement Spread
6	.05	−.05	−.15	.15	.5	.5	1	1	200	Asymmetry
7	.1(.55)	−.1(.45)	−.1(.45)	.1(.55)	.25	.75	1	1	1500	Stimulus Frequency
8	.1	−.1	−.1	.1	.25	.75	1	1	200	Asymmetry
9	.1(.55)	−.1(.45)	−.1(.45)	.1(.55)	.5	.5	.5	1.5	1500	Stimulus Intensity
10	.1	−.1	−.1	.1	.5	.5	.5	1.5	200	Asymmetry
11	−.4(.3)	−.8(.1)	.3(.65)	.9(.95)	.25	.75	.5	1.5	3000	Combined Asymmetries
12	−.65	−.75	.55	.85	.25	.75	.5	1.5	2000	

each facet of the table is discussed below.

As in Chapter II, all tasks are binary-action tasks ($y[t]=1$ or $y[t]=0$). On each time step t the environment sends to the learning system one of two stimuli, represented as vectors, and which are denoted $\vec{x}^1$ and $\vec{x}^2$. For Tasks 1–8,

$$\vec{x}^1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \quad \text{and} \quad \vec{x}^2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}. \tag{10}$$

For Tasks 9–12,

$$\vec{x}^1 = \begin{pmatrix} .5 \\ .5 \\ 0 \end{pmatrix} \quad \text{and} \quad \vec{x}^2 = \begin{pmatrix} 0 \\ 1.5 \\ 1.5 \end{pmatrix}. \tag{11}$$

In both cases the two stimuli are clearly distinguishable via the first and third components, but are similar via the second component.

One difference between the stimuli of Tasks 1–8 and those of Tasks 9–12 is their intensity. The two stimuli of Tasks 1–8 are equally intense, whereas in Tasks 9–12 the first stimulus is half as intense as the stimuli of the other tasks, and the second stimulus is one-and-a-half times as intense. This difference is indicated in Table 3 in the columns labeled "Stimulus Intensity."

On all tasks the stimulus presented is selected probabilistically. On Tasks 1–6 and 9–10, the two stimuli are presented with equal frequency, i.e., each with a probability of .5. On any given time step of Tasks 7–8 and 11-12, $\vec{x}^2$ is presented with a probability of .75 and $\vec{x}^1$ with a probability of .25. These probabilities are listed in Table 3 in the columns labeled "Stimulus Frequency."

The odd-numbered tasks are binary-reinforcement tasks and the even-numbered tasks are continuous-reinforcement tasks. In the binary-reinforcement tasks, the reinforcement r received from the environment is either a $+1$ (success) or a -1 (failure), whereas in the continuous-reinforcement tasks the reinforcement can take on any real value.

In all tasks, reinforcement $r[t+1]$ is a stochastic function of the preceding action $y[t]$ and stimulus $\vec{x}[t]$. Since there are two possible stimuli and two possible actions, there are four possible combinations of stimulus and action. Columns 3–6 of Table 3 list the expected value of reinforcement for each task for each of these four combinations.

For the continuous-reinforcement tasks, reinforcement is computed from the

expected values in Table 3 as

$$r[t+1] = r_{y[t]}^{\vec{x}[t]} + \eta[t],$$

where $r_y^{\vec{x}}$ denotes the expected value of reinforcement corresponding to stimulus $\vec{x}$ and action y, and where $\eta[t]$ is a normally-distributed random variable of mean 0 and standard deviation σ_r. For all tasks except Task 12, $\sigma_r = .1$. For Task 12, $\sigma_r = .025$.

For the binary-reinforcement tasks, reinforcement is computed from the probabilities given in parentheses in Columns 3–6 of Table 3. Letting $P_y^{\vec{x}}$ denote $P\{r[t+1] = 1 \mid \vec{x}[t] = \vec{x}, y[t] = y\}$, i.e., the probability of a successful outcome given stimulus $\vec{x}$ and action y, the expected values given in Table 3 are computed as follows:

$$r_y^{\vec{x}} = 1 \cdot P_y^{\vec{x}} - 1 \cdot (1 - P_y^{\vec{x}}).$$

Simulation runs involving different tasks ran for different numbers of steps. For each task, the number of steps its runs ran is listed in Table 3 in the column labeled "Steps."

Finally, the last column of Table 3 provides a verbal description of the type of problem each pair of tasks presents to a learning algorithm. These descriptions are elaborated later in this chapter.

Algorithms

All 11 algorithms use the linear-mapping approach to associative learning. That is, they all update an action-association vector $\vec{w}[t] = (w_1[t], w_2[t], \ldots, w_n[t])$ and select their actions according to (7) and (8) with $\sigma_y = .3$ for all algorithms.

The update rules of some algorithms involve $\pi[t]$, the probability that $y[t] = 1$ given $\vec{x}[t]$. Determining $y[t]$ according to (7) implies that

$$\pi[t] = \Phi(s[t]/\sigma_y),$$

where Φ is the unit normal distribution function. All normally-distributed random numbers used in these experiments were approximated by linear interpolation from tables and uniformly-distributed random numbers as described in Chapter II.

The learning algorithms studied in this chapter differ only in the rules they use to update the weight vector $\vec{w}[t]$. The update rules and other relevant equations for the 11 algorithms used in these experiments are given in Table 4. The update rules are all simple extensions to the case of associative learning of the update rules discussed in Chapter II. For the special case in which the same stimulus vector is presented on every time step, and in which that vector has only one component, whose value is 1, Algorithms 4-9 reduce exactly to the like-numbered algorithms of Chapter II. Algorithm 4 is closely related to algorithms previously studied by Farley and Clark (1954) and also by my colleagues and I (Barto, Sutton and Brouwer, 1981; Barto and Sutton, 1981).

The update rules of Algorithms 4–7 of this chapter are very similar to those of Algorithms 4–7 of Chapter II (compare Tables 2 and 4). The primary difference (besides the fact that the new rules update all the components $w_i[t]$ of a vector $\vec{w}[t]$ whereas the rules of Chapter II update a scalar $w[t]$) is that the new update rules each have one more factor than the corresponding update rules of Chapter II. In all cases the additional factor determining the modification of $w_i[t]$ is the corresponding stimulus-vector component $x_i[t]$.

In the algorithms of Chapter II, $w[t]$ directly effects $y[t]$. In the algorithms of this chapter, each component $w_i[t]$ of $\vec{w}[t]$ has an effect (via (7) and (8)) on $y[t]$ proportional to the value of $x_i[t]$, the corresponding stimulus-vector component.

Table 4. Associative Learning Algorithms.

Algorithm	Update Rule	Relevant Tasks
1	$w_i[t+1] = w_i[t] + \begin{cases} \alpha(y[t] - \pi[t])x_i[t], & \text{if r[t+1]=1} \\ 0, & \text{if r[t+1]=-1} \end{cases}$	1,3,5,7,9,11
2	$w_i[t+1] = w_i[t] + \begin{cases} 0, & \text{if r[t+1]=1} \\ \alpha(1 - y[t] - \pi[t])x_i[t], & \text{if r[t+1]=-1} \end{cases}$	1,3,5,7,9,11
3	$w_i[t+1] = w_i[t] + \begin{cases} \alpha(y[t] - \pi[t])x_i[t], & \text{if r[t+1]=1} \\ \alpha(1 - y[t] - \pi[t])x_i[t], & \text{if r[t+1]=-1} \end{cases}$	1,3,5,7,9,11
4	$w_i[t+1] = w_i[t] + \alpha r[t+1](y[t] - \frac{1}{2})x_i[t]$	1-12
5	$w_i[t+1] = w_i[t] + \alpha r[t+1](y[t] - \pi[t])x_i[t]$	1-12
6	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - r[t])(y[t] - \frac{1}{2})x_i[t]$	3,4
7	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - r[t])(y[t] - \pi[t])x_i[t]$	3,4
8	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - p[t])(y[t] - \frac{1}{2})x_i[t]$	1-12
9	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - p[t])(y[t] - \pi[t])x_i[t]$	1-12
10	$w_i[t+1] = w_i[t] + \begin{cases} \alpha(y[t] - \frac{1}{2})x_i[t], & \text{if r[t+1]=1} \\ 0, & \text{if r[t+1]=-1} \end{cases}$	1,3,5,7,9,11
11	$w_i[t+1] = w_i[t] + \begin{cases} 0, & \text{if r[t+1]=-1} \\ \alpha(1 - y[t] - \frac{1}{2})x_i[t], & \text{if r[t+1]=1} \end{cases}$	1,3,5,7,9,11

Where:

$$w_i[0] = 0, \qquad v_i[0] = 0, \qquad y[t] \in \{1, 0\}, \qquad \alpha > 0,$$

$\pi[t]$ is the probability that $y[t] = 1$, given $\vec{x}[t]$,

$$y[t] = \begin{cases} 1, & \text{if } s[t] + \eta[t] > 0; \\ 0, & \text{otherwise,} \end{cases}$$

where $\eta[t]$ is a normally distributed random variable of mean 0 and standard deviation $\sigma_y = .3$,

$$s[t] = \sum_{i=1}^{n} w_i[t]x_i[t] \qquad \text{and} \qquad p[t] = \sum_{i=1}^{n} v_i[t]x_i[t].$$

The components of the stimulus vector, then, give a measure of how each component of the action-association vector affects the action. If a stimulus component is zero, then changing the corresponding w_i will have no affect on the action. If a stimulus component x_i is large, then a modification of w_i will have a large influence on the action selected. This is why the multiplicative $x_i[t]$ term appears in the new update rules. As a result of this term, each $w_i[t]$ is changed in proportion to the size of the resultant influence on the action selected.

The update rule of each algorithm includes a product of two terms, one depending on the action selected, either $y[t]-\frac{1}{2}$, $y[t]-\pi[t]$, $1-y[t]-\frac{1}{2}$, or $1-y[t]-\pi[t]$, and one depending on the presence or absence of a stimulus component, $x_i[t]$. This product, denoted in later chapters as $e_i[t]$, is called the ***eligibility*** (after Klopf, 1972, 1982) of w_i because it indicates the extent to which w_i is eligible for undergoing modification should reinforcement be received.

Algorithms 1–3 of this chapter also correspond to the like-numbered algorithms of Chapter II. These algorithms, however, underwent more radical change in being converted to an associative-learning form because they are expressed (in Chapter II) as changes in $\pi[t]$ rather than as changes in $w[t]$. In their new forms they directly determine changes in $\vec{w}[t]$.

Algorithms 10 and 11 do not correspond to any algorithms discussed in Chapter II. In this and the previous chapter, pairs of algorithms are compared that differ only in that one uses $y[t]-\frac{1}{2}$ and the other uses $y[t]-\pi[t]$ in their update rules. Given the modifications described above of Algorithms 1–3, all of which use $y[t]-\pi[t]$, it is possible also to include the corresponding algorithms that use $y[t]-\frac{1}{2}$. Algorithms 10 and 11 correspond to Algorithms 1 and 2 in this way. There is no need to add an algorithm for the $y[t]-\frac{1}{2}$ case corresponding to Algorithm 3 because Algorithm 4 already serves this function.

Reinforcement Comparison in Associative Learning. Algorithms 6 and 7 are simple reinforcement-comparison algorithms very similar to Algorithms 6 and 7 of Chapter II. These algorithms compare the current reinforcement with the immediately-preceding reinforcement. Although this technique worked well on the tasks of Chapter II, there are reasons to doubt its success on associative-learning tasks. It is much more difficult for an algorithm to compare current with past reinforcement levels properly on associative tasks than it is on nonassociative tasks. The problem is that for many associative tasks it is possible to obtain high reinforcement levels in the presence of one stimulus and only low reinforcement levels in the presence of another. Tasks 3 and 4, for example, are of this sort. On such tasks, if different stimuli occur on two successive steps, then the change in reinforcement may be primarily due to the change in stimulus rather than to the action selected by the learning algorithm. In such cases the use of $r[t+1] - r[t]$ (as in Algorithms 6 and 7) would be inappropriate and would mislead the learner. To illustrate this problem while minimizing the complexity of the experiments, Algorithms 6 and 7 were applied only to Tasks 3 and 4.

To obtain the maximum advantage from a reinforcement-comparison technique in associative learning, reinforcement received on the current step should be compared with reinforcement received on previous steps *on which the same stimulus occurred.* How are past reinforcement levels for different stimuli to be recorded and kept separate from each other? This problem is particularly difficult if one assumes, as is done here, that stimuli are not individually recognizable and separable (as in the independent-associations approach).

One way of solving this problem is to view the sequence of stimuli, and the reinforcement levels obtained by acting in response to them, as a training sequence for supervised learning pattern classification. To this end, Algorithms 8 and 9 use a version of the Widrow-Hoff, or Adaline, algorithm (Widrow and Hoff, 1960; Widrow, 1962): A modifiable parameter vector $\vec{v}[t] = (v_1[t], v_2[t], ..., v_n[t])$ is re-

quired in addition to $\vec{w}[t]$. Whereas $\vec{w}[t]$ maps a stimulus to an action, $\vec{v}[t]$ maps a stimulus to an average of the reinforcement levels obtained when the stimulus (or similar stimuli) occurred in the past. $\vec{v}$ is called the ***reinforcement-association vector*** (whereas $\vec{w}$ is called the ***action-association vector***), and this average of past reinforcement levels is called the ***predicted reinforcement*** and denoted $p[t] \in \Re$. It is computed from the current stimulus $\vec{x}[t]$ in the usual way for a linear-mapping approach to association:

$$p[t] = \sum_{i=1}^{n} v_i[t] x_i[t]. \tag{12}$$

$\vec{v}[t]$ is updated so that $p[t]$ becomes an average of the appropriate past reinforcement levels. The following equation changes $\vec{v}[t]$ according to the discrepancy between the predicted reinforcement $p[t]$ and the corresponding actual reinforcement $r[t+1]$:

$$v_i[t+1] = v_i[t] + \beta(r[t+1] - p[t])x_i[t], \tag{13}$$

for $t = 0, 1, \ldots,$ $i = 1, \ldots, n$, $v_i[0] = 0$ and $\beta = .1$ for all simulations discussed in this chapter. The discrepancy between expected reinforcement $p[t]$ and actual reinforcement $r[t+1]$, in addition to being the error in the predicted reinforcement, is also the quantity needed as a comparison of the current reinforcement level with past reinforcement levels associated with the current stimulus. Algorithms 8 and 9 use this discrepancy in this way. For example, Algorithm 8 updates its action-association vector $\vec{w}[t]$ as follows:

$$w_i[t+1] = w_i[t] + \alpha\,(r[t+1] - p[t])(y[t] - \frac{1}{2})x_i[t],$$

for $t = 0, 1, \ldots$ and $i = 1, \ldots, n$. When the actual reinforcement $r[t+1]$ is greater (less) than the predicted reinforcement $p[t]$, w_i is changed so as to make the action selected more (less) likely as a response to the stimulus. An associative reinforcement-learning algorithm that remembers and compares reinforcement levels in this way has previously been investigated by my colleagues and I (Barto,

Sutton and Brouwer, 1981). This method is a generalization of that used in Chapter II. For the special case in which the same stimulus is presented on every time step, and in which that vector has only one component, whose value is 1, (12) and (13) reduce exactly to (3), and Algorithms 8 and 9 of this chapter reduce exactly Algorithms 8 and 9 of Chapter II.

Although a reinforcement-comparison algorithm for associative learning is necessarily more complex than those considered so far, such an algorithm could potentially perform much better than non-reinforcement-comparison algorithms. The results reported in Chapter II suggest that reinforcement-comparison algorithms learn much more rapidly than non-reinforcement-comparison algorithms on nonassociative learning tasks with unbalanced reinforcement. It seems likely, therefore, that reinforcement-comparison algorithms might have a similar advantage in associative-learning tasks. The results of the experiments of this chapter bear out this hypothesis.

Results

For each task and algorithm, simulation runs were made for many values of learning-rate parameter α. For Tasks 1–10, the α values were powers of two from 2^1 to 2^{-11}. For Tasks 11 and 12, the α values were powers of two from 2^1 to 2^{-15}. For each task, algorithm, and α, 100 simulation runs were made each differing only in the initial seed for the random-number generator. The durations of the runs varied from task to task as listed in the column of Table 3 labeled "Steps." At the end of each run the final parameter vector $\vec{w}[101]$ was recorded. From this, the probability π^i of the learning system selecting Action 1 on the next step, had the run continued and stimulus i been presented, was computed as

follows:

$$\pi^i = P\left\{y[101] = 1 \mid x[101] = \vec{x}^i\right\} = \Phi\left(\sum_{j=1}^{3} w_j[101]x_j^i/\sigma_y\right),$$

where $P\{\cdot \mid \cdot\}$ denotes a conditional probability. Using this, the known stimulus presentation frequencies, and the expected values of the reinforcement as a function of stimulus and action (listed in Table 3), one can compute the expected value of the reinforcement on the next step, had the run continued:

$$E\{r[101]\} = p^1\pi^1 r_1^1 + p^1(1-\pi^1)r_0^1 + p^2\pi^2 r_1^2 + p^2(1-\pi^2)r_0^2,$$

where p^i is the probability of stimulus $\vec{x}^i$ being presented, $i = 1, 2$, and r_j^i is the expected value of the reinforcement given that stimulus i and action j have occurred. These probabilities and expected values depend on the task as listed in Table 3.

The expected value of reinforcement on the step following the end of a run is a good measure of how well the learning algorithm has learned on the run. If it has learned rapidly and correctly, $\vec{w}$ will have reached a value for which the expected value of reinforcement is high. Figures 9–20 are plots of the expected value of reinforcement at the end of the runs, averaged over the 100 runs with each task, algorithm and learning-rate parameter value. Each figure presents all the data from a single task. The figures presenting the data from the binary-reinforcement tasks include two graphs per figure. The lower graph of these figures presents the data for that task from Algorithms 1–3 and 10–11 while the upper graph presents the data from the other algorithms run on the task, plus a re-plot of the data for Algorithm 1. For each algorithm a line is plotted connecting the performances of the algorithm at different values of the learning-rate parameter α.

The general pattern to these plots is the same as that in Chapter II. In most cases the performance of each algorithm on each task is an inverted-U shaped function of the learning-rate parameter α. Some algorithms, particularly the very

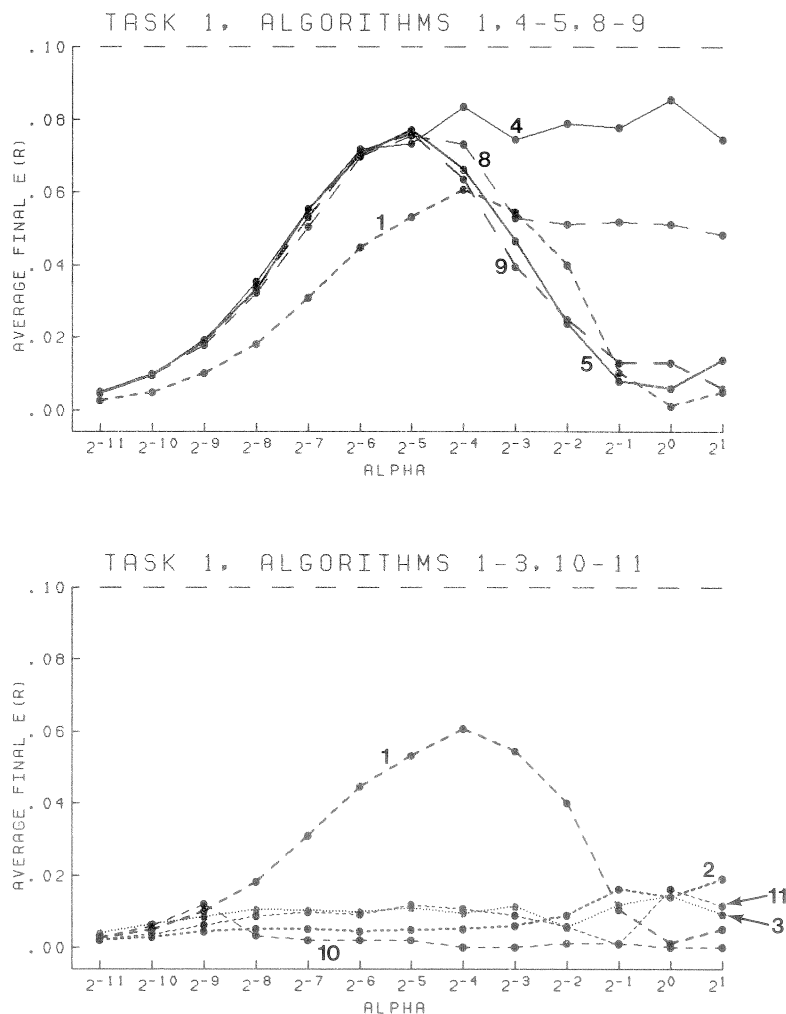


Figure 9. Algorithm Performance on Task 1 of Chapter III. Task 1 is a binary-reinforcement task with *no asymmetries*. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

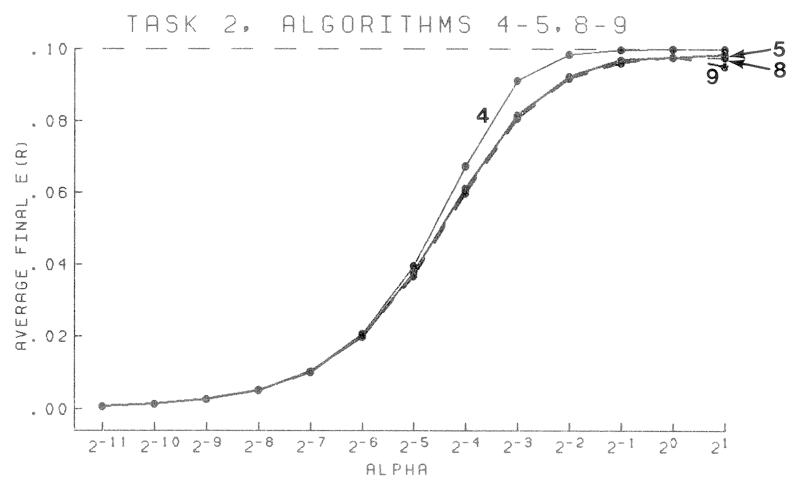


Figure 10. Algorithm Performance on Task 2 of Chapter III. Task 2 is a continuous-reinforcement task with *no asymmetries.* Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

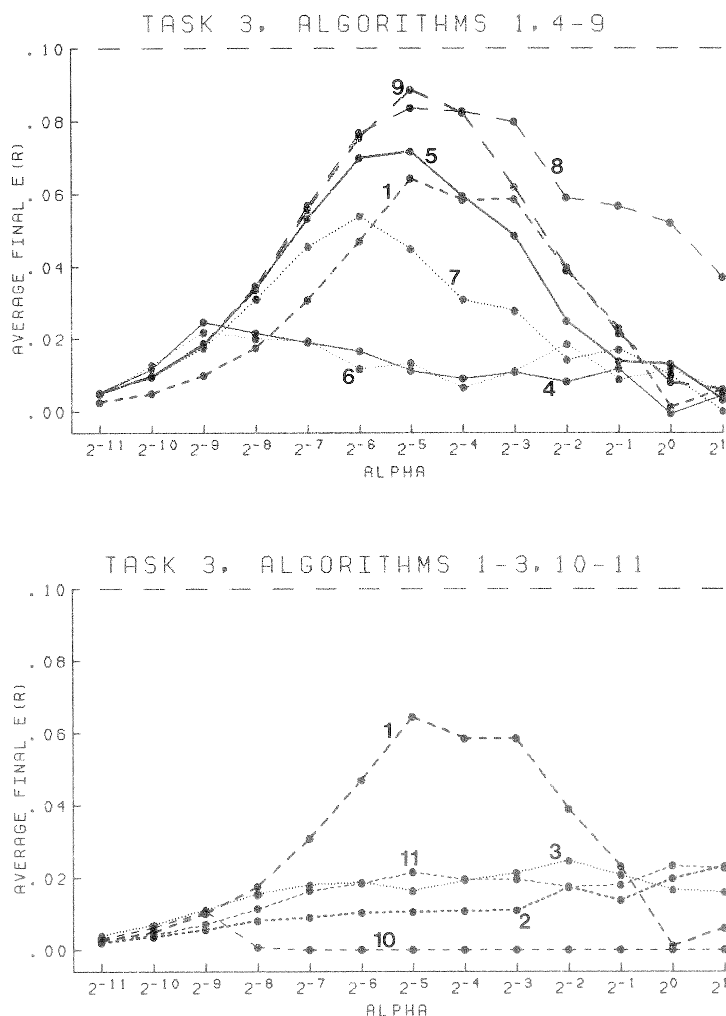


Figure 11. Algorithm Performance on Task 3 of Chapter III. Task 3 is a binary-reinforcement task with *reinforcement-level asymmetry.* Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

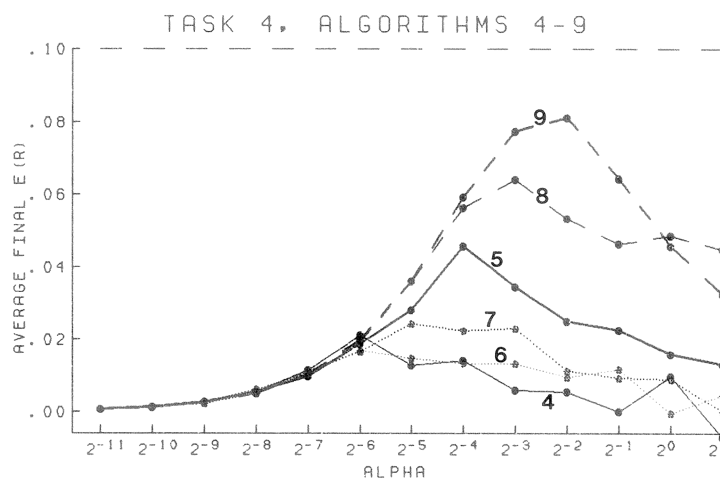


Figure 12. Algorithm Performance on Task 4 of Chapter III. Task 4 is a continuous-reinforcement task with *reinforcement-level asymmetry*. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

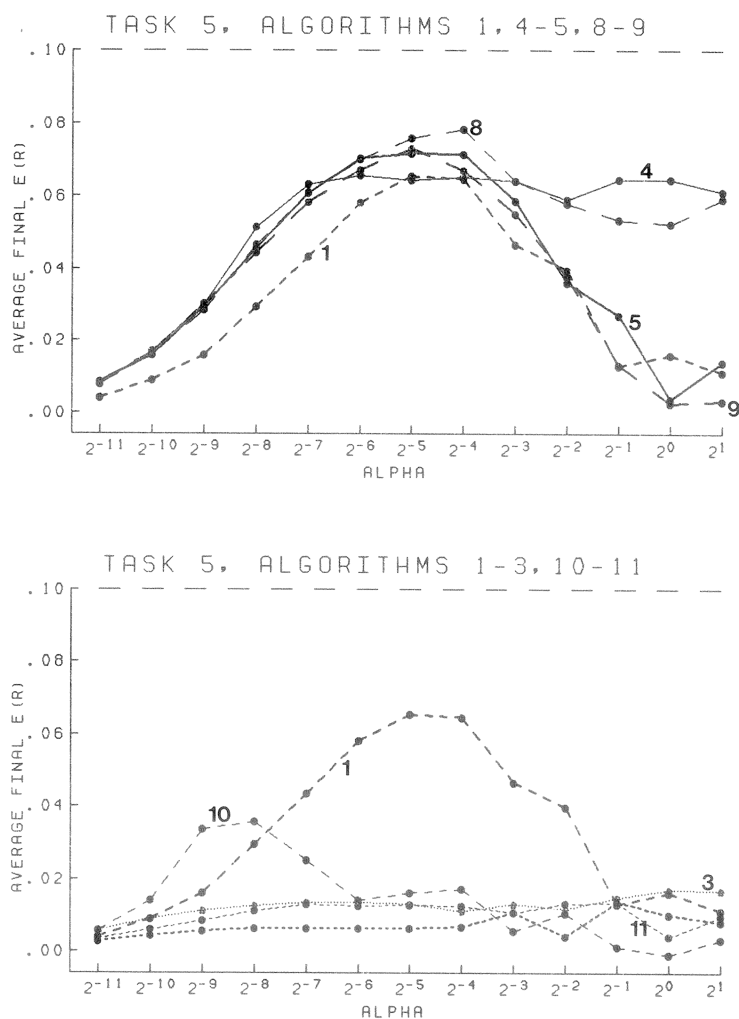


Figure 13. Algorithm Performance on Task 5 of Chapter III. Task 5 is a binary-reinforcement task with *reinforcement-spread asymmetry.* Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

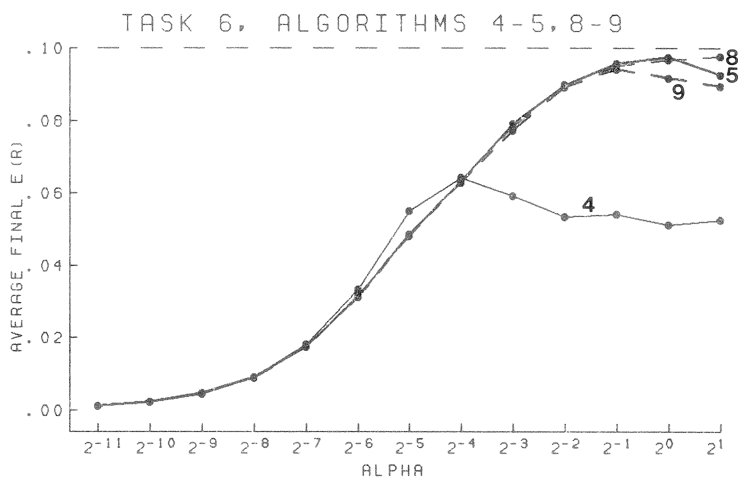


Figure 14. Algorithm Performance on Task 6 of Chapter III. Task 6 is a continuous-reinforcement task with ***reinforcement-spread asymmetry***. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

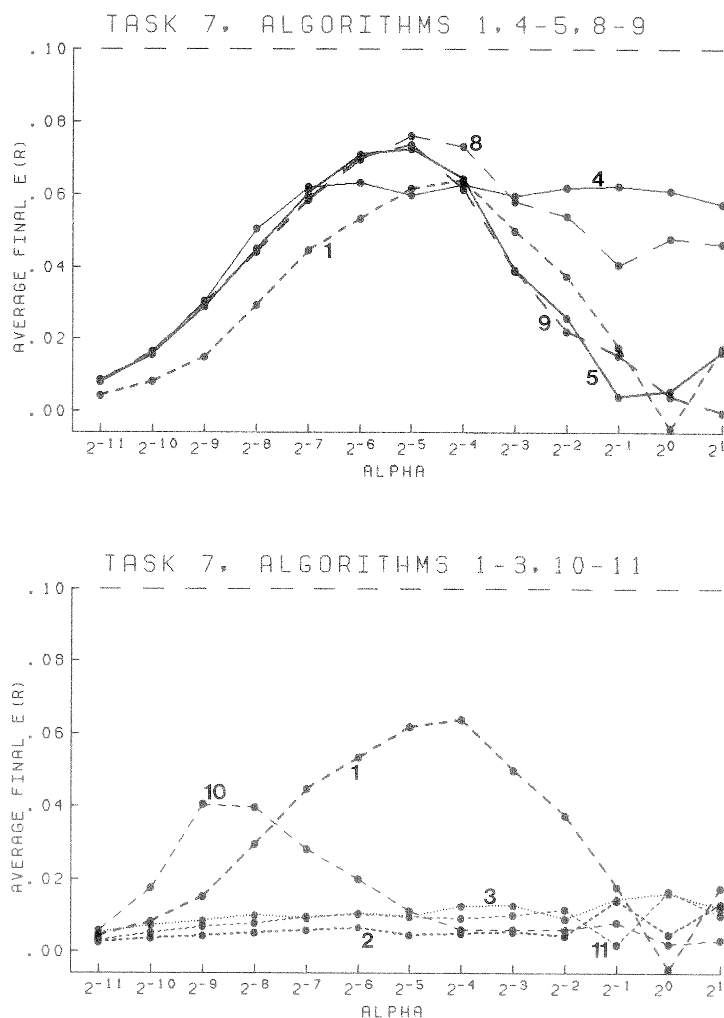


Figure 15. Algorithm Performance on Task 7 of Chapter III. Task 7 is a binary-reinforcement task with ***stimulus-frequency asymmetry.*** Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

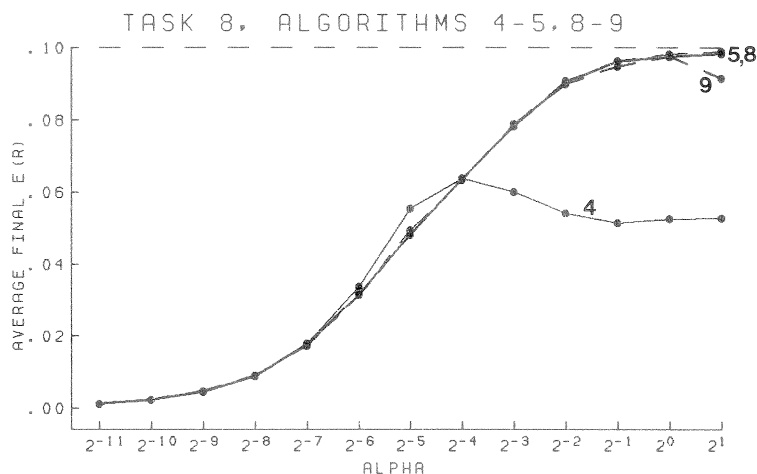


Figure 16. Algorithm Performance on Task 8 of Chapter III. Task 8 is a continuous-reinforcement task with ***stimulus-frequency asymmetry.*** Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

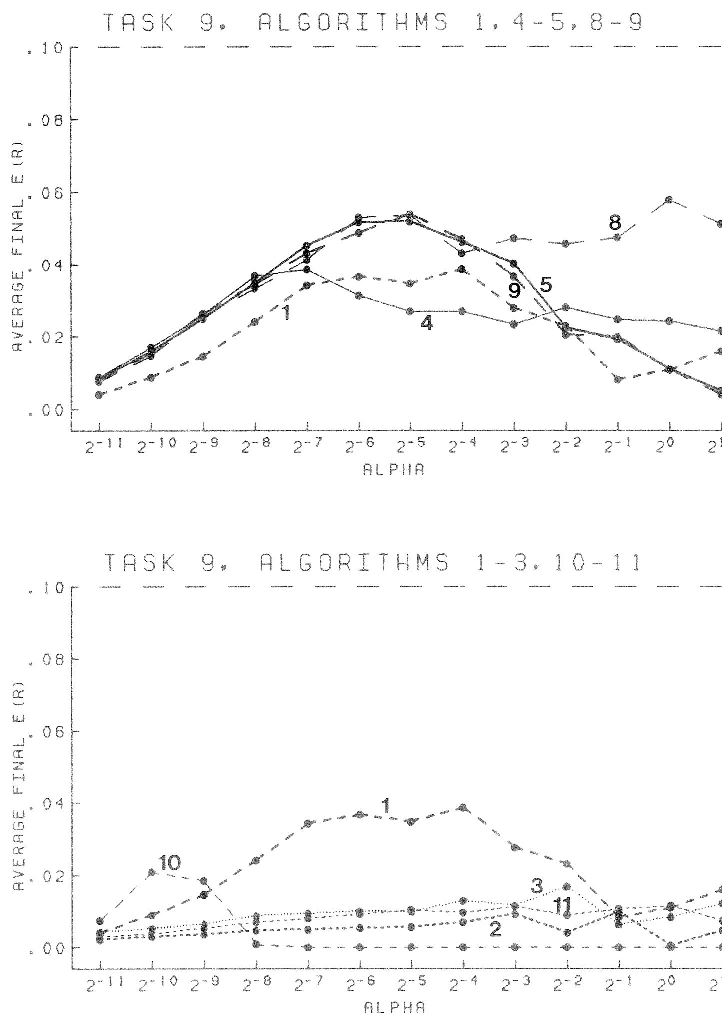


Figure 17. Algorithm Performance on Task 9 of Chapter III. Task 9 is a binary-reinforcement task with ***stimulus-intensity asymmetry***. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

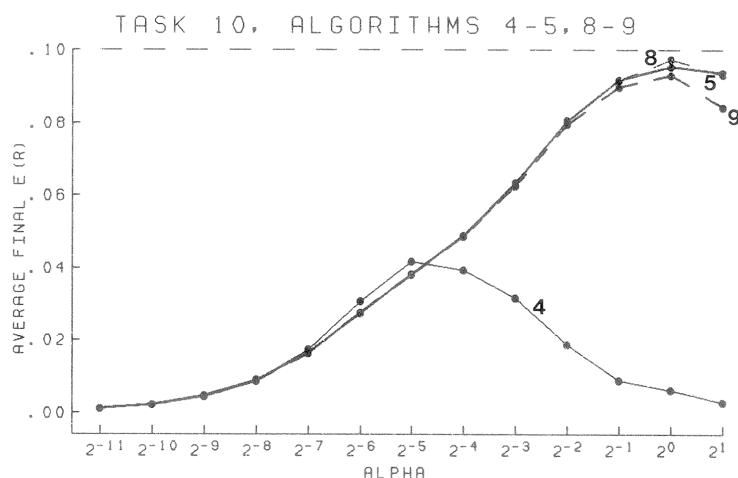


Figure 18. Algorithm Performance on Task 10 of Chapter III. Task 10 is a continuous-reinforcement task with ***stimulus-intensity asymmetry.*** Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

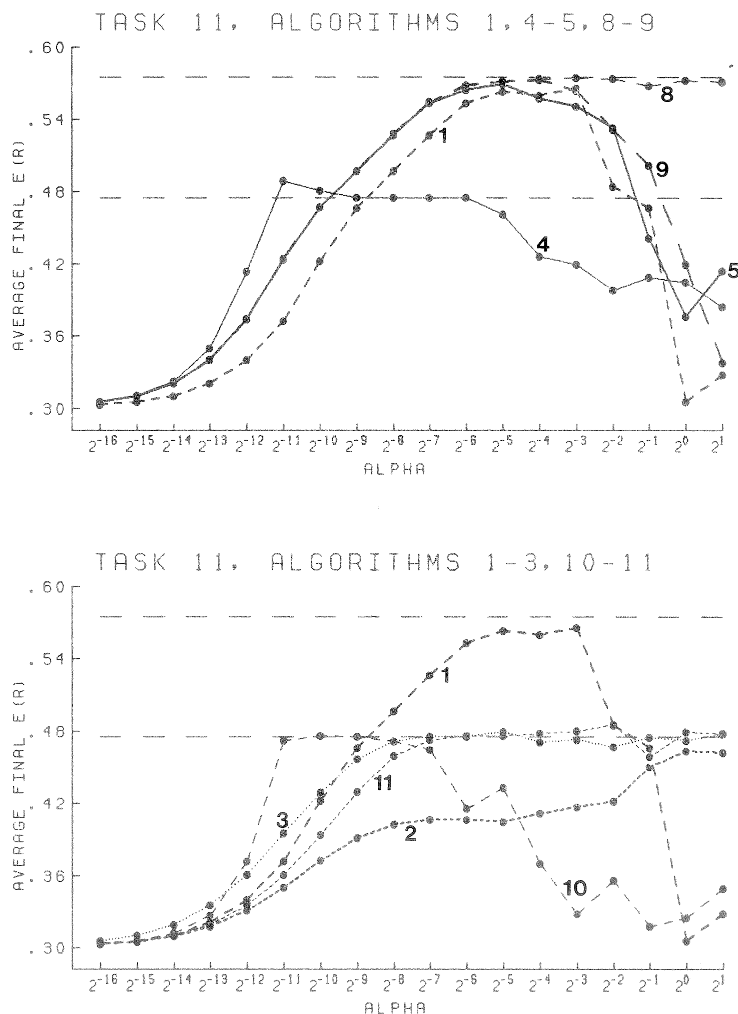


Figure 19. Algorithm Performance on Task 11 of Chapter III. Task 11 is a binary-reinforcement task with *combined asymmetries.* Runs with this task lasted many steps so that performance would near its asymptotic value. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The upper horizontal dashed line indicates the optimal performance level, and the lower one indicates the highest performance level achievable without discriminating between stimuli.

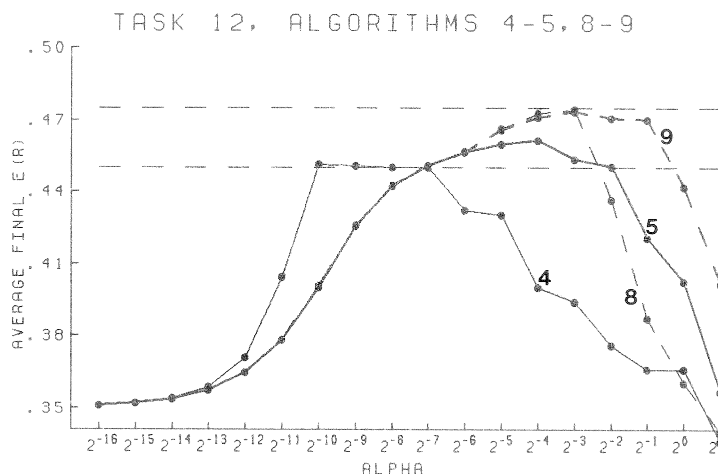


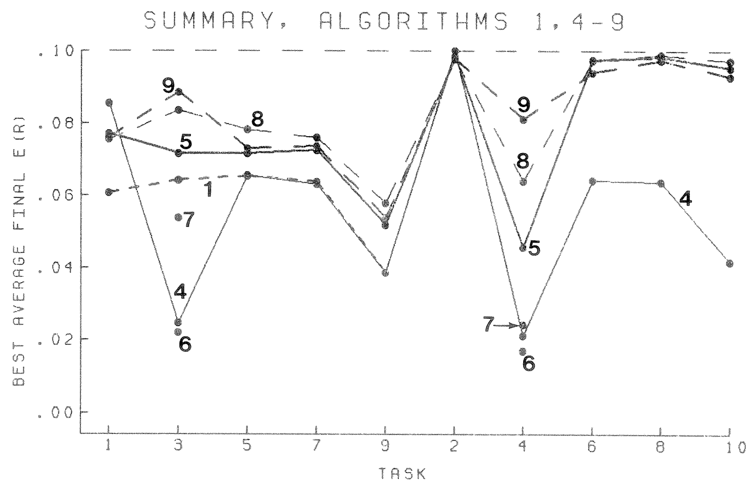
Figure 20. Algorithm Performance on Task 12 of Chapter III. Task 12 is a continuous-reinforcement task with *combined asymmetries.* Runs with this task lasted many steps so that performance would near its asymptotic value. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The upper horizontal dashed line indicates the optimal performance level, and the lower one indicates the highest performance level achievable without discriminating between stimuli.

poorly performing algorithms, such as Algorithms 3, 2, and 11, in some cases appear to continue to improve in performance as α is increased. A closer look at the data and other experiments with higher values of α (not reported here) suggest that, rather than consistently improving, these algorithms merely become more erratic in their performance if α is increased beyond the values used here. The performance of Algorithms 2, 3, and 11 never approaches the performance levels of Algorithms 1, 4, 5, 8, or 9.

In a few cases something similar happened with the better performing algorithms—performance became erratic without becoming plainly poorer at high α values. In two cases—Algorithm 4 on Task 1, and Algorithm 8 on Task 9 — the performance at a very high value of α exceeded that at more moderate levels where performance was more consistent. These two data points are probably over-estimates of the performance of these algorithms on these tasks. Fortunately, however, the suspect data points are not so different from the others as to influence the statistical significance of any result. No adjustments were made for the suspect data points in either Figure 21 or the tests of statistical significance.

As with the data of Chapter II, one good way to interpret these data is to compare the algorithms' *best* performances on each task, i.e., their performances with the α values that resulted in the highest performance levels. Using this measure, Figure 21 summarizes the data from Tasks 1–10, showing the best performances of all algorithms on all tasks.

Tests of Statistical Significance. A test of statistical significance was performed on all differences between the best performance levels of all algorithms on all tasks. Each best performance is an average over 100 runs of the performance of a particular algorithm on a particular task at a particular α value. To determine which differences between averages are statistically significant, a one-tailed t-test



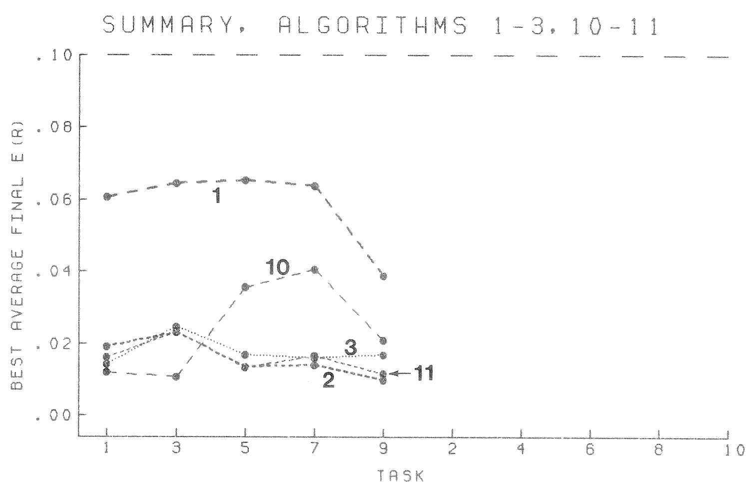


Figure 21. Summary of Algorithm Performance on Tasks 1–10 of Chapter III. Each point represents the performance level of a particular algorithm on a particular task with the α value at which performance was best for that algorithm on that task. Points due to the same algorithm (on different tasks) are connected by lines. The number labeling each line indicates the associated algorithm. The horizontal dashed line indicates the optimal performance level.

was applied to each pair of averages, as discussed in Chapter II. Table 5 lists by task all statistically significant differences between the best average performances.

Table 5. Statistical Significance Results of Chapter III.

Task	Statistically Significant Performance Orderings of Algorithms
1	$4,5,8,9 \gg 1 \gg 2,3,10,11; \quad 4 > 5,8,9$
2	$4 \gg 8,9; \quad 1 \gg 10 \gg 2,11$
3	$9 \gg 8 \gg 5 \gg 1 > 7 \gg 4 \gg 6; \quad 2,3,11 \gg 10$
4	$9 \gg 8 \gg 5 \gg 4,6,7; \quad 7 > 6$
5	$8 \gg 1,4; \quad 8 > 5; \quad 5,9 > 1,4 \gg 10 \gg 2,3,11$
6	$5,8 \gg 9 \gg 4$
7	$8,9 \gg 1,4 \gg 10 \gg 2,3,11; \quad 5 \gg 4; \quad 5 > 1$
8	$5,8,9 \gg 4; \quad 8 > 9;$
9	$5,8,9 \gg 1,4 \gg 10 \gg 2,11$
10	$8 \gg 9; \quad 5 > 5; \quad 5,8,9 \gg 4$
11	$8 \gg 1,5; \quad 9 > 5; \quad 9 \gg 1; \quad 1,5,8,9 \gg 2,3,4,10,11; \quad 11 > 10$
12	$8 \gg 9 \gg 5 \gg 4$

Where $\gg$ denotes an ordering significant at the $P < .01$ level, and $>$ denotes an ordering significant at the $P < .05$ level.

Discussion

The 12 tasks were chosen to investigate the problems of stimulus discrimination in the face of an asymmetrical treatment of stimuli. For a particular task and stimulus, the preferable (correct) action is the one with the highest expected value of reinforcement, as given in Table 3. On all tasks, the correct action when stimulus $\vec{x}^1$ is presented differs from that when stimulus $\vec{x}^2$ is presented. To maximize reinforcement, a learning algorithm must discriminate between, and respond differently to, the two stimuli. For simplicity, all tasks were constructed so that Action 1 is the best action when stimulus $\vec{x}^1$ is presented, and Action 0 is the best action when stimulus $\vec{x}^2$ is presented.*

For all tasks the two stimulus vectors $\vec{x}^1$ and $\vec{x}^2$ (defined by (10) and (11)) are similar and yet distinguishable. $\vec{x}^1$ and $\vec{x}^2$ are similar by virtue of their second components, which are both positive, and distinguishable by virtue of their first and third components, which are positive for one stimulus and zero for the other. Since similar stimuli must be responded to differently, these tasks all involve *misleading generalization.*

When $\vec{x}^1$ occurs, the first and second weight vector components are modified. When $\vec{x}^2$ occurs, the second and third weight vector components are modified. Since in the first case Action 1 is correct, and in the second, Action 0 is correct, the first weight vector component w_1 becomes positive (associated with Action 1), and the third weight vector component w_3 becomes negative (associated with Action 0), but it is unclear what happens to the second weight vector component w_2. Ideally, the countervailing influences on w_2 cancel out, leaving it near zero. This result is ideal because, by (7) and (8), it leaves the action in response to $\vec{x}^1$ to be determined by w_1 and the action in response to $\vec{x}^2$ to be determined by

* The learning algorithms did not "know" which stimulus had which number, so there was no way they could take advantage of this uniformity.

w_3. This result is most likely to occur on tasks that are symmetrical with respect to the two stimuli. True symmetry guarantees that the countervailing influences on w_2 are of equal strength, and thus gives the greatest likelihood that they will cancel out.

The tasks of this chapter were designed in pairs, 1 with 2, 3 with 4, etc. (each pair consisting of one binary-reinforcement and one continuous-reinforcement task). The first pair of tasks, Tasks 1 and 2, were designed to be completely symmetrical with respect to the two stimuli. The next 4 pairs, 3-4, 5-6, 7-8, and 9-10, were each designed to treat the two stimuli asymmetrically in one and only one way. Finally, the tasks of the pair 11-12 combined all 4 forms of asymmetry.

The continuous-reinforcement tasks used in these experiments are significantly easier than the binary-reinforcement tasks. For this reason, the experiments involving continuous-reinforcement tasks were run for fewer steps than those involving binary-reinforcement tasks (200 steps as opposed to 1500). If both sorts of experiments had been run for 1500 steps, performance differences between some algorithms on the easier tasks would have been lost in a ceiling effect, as many algorithms would have performed near optimally.

In the following discussion, references to particular performance differences between algorithms as being either "significant" or "not significant" are referring to statistical significance at the $P < .05$ probability criterion, unless otherwise noted. More information on the significance level of each result, and on the statistical test used, can be found in the "Results" section.

Tasks 1 and 2: No Asymmetries. Tasks 1 and 2 are completely symmetrical with respect to the two stimuli. On these tasks, the stimuli are presented with equal frequency and equal intensity. The expected values of reinforcement associated with the two stimuli are equal (although which action has the higher expected

value switches from one stimulus to the other). As discussed above, this symmetry makes these tasks the *least* likely to be influenced by the problems of misleading generalization between similar stimuli.

Tasks 1 and 2 are closely related to Tasks 3 and 6 of Chapter II. These 4 tasks have similar distributions of expected value of reinforcement as a function of the action selected. On all 4 of these tasks the expected value of reinforcement is positive if one action is chosen and negative, by the same amount, if the other action is chosen. For such tasks the reinforcement distribution is said to be balanced, i.e., its expected value as a function of action is distributed symmetrically around the neutral value zero.

The similarity between Tasks 1 and 2 of this chapter and Tasks 3 and 6 of Chapter II is born out in similar performances of corresponding algorithms on corresponding tasks (compare Figures 8 and 21). In Chapter II, Algorithms 4–9 all performed essentially perfectly on the continuous-reinforcement task (Task 6) and here the closely related Algorithms 4–9 also all performed essentially perfectly on the corresponding task (Task 2). The binary-reinforcement Task 3 of Chapter II corresponds to Task 1 of the present chapter. On both tasks, Algorithm 4 performed best, followed by Algorithms 5, 8, and 9, all of which performed better that Algorithm 1, which in turn performed better than Algorithms 2 and 3. The two new algorithms added in this chapter, Algorithms 10 and 11, fell in with the last group of poorest performing algorithms.

Tasks 3 and 4: Reinforcement-Level Asymmetry. On Tasks 3 and 4, when stimulus $\vec{x}^2$ occurs, the expected value of reinforcement is always high, and when stimulus $\vec{x}^1$ occurs, the expected value of reinforcement is always low. These tasks differ from Tasks 1 and 2 only in that the expected value of the reinforcement is unbalanced in this way. This asymmetry in the treatment of stimuli is called

reinforcement-level asymmetry. Nonassociative tasks that differ from each other in a similar way are discussed in Chapter II as *unbalanced-reinforcement tasks*. Ignoring for the moment generalization between stimuli, Tasks 3 and 4 of this chapter are much like a combination of Tasks 1 and 2, and 4 and 5, respectively, of Chapter II.

In the experiments of Chapter II, Algorithm 4 performed the best of all algorithms on the balanced-reinforcement tasks (Tasks 3 and 6) and poorly on the unbalanced-reinforcement tasks (Tasks 1, 2, 4, and 5). Algorithm 4's performance in this chapter is similar. The upper graph of Figure 21 shows that whereas Algorithm 4 performed best of all algorithms on the balanced-reinforcement tasks (Tasks 1 and 2) of this chapter, it performed very poorly on the tasks with reinforcement-level asymmetry (Tasks 3 and 4), which are unbalanced-reinforcement tasks. Algorithm 5's performance is also similar to its performance in Chapter II. In both cases, Algorithm 5 performed about as well as Algorithms 8 and 9 on the balanced-reinforcement tasks and and significantly worse on the unbalanced-reinforcement tasks. Overall the conclusions of Chapter II regarding reinforcement-comparison mechanisms are confirmed: these mechanisms, at least as implemented in Algorithms 8 and 9, significantly improve performance on unbalanced-reinforcement tasks, and do not degrade performance on balanced-reinforcement tasks.

On Tasks 3 and 4 the expected value of reinforcement varies more with the stimulus presented than with the action selected. This reinforcement-level asymmetry poses a particular challenge to reinforcement-comparison algorithms. Reinforcement-comparison algorithms that compare successive reinforcement levels without regard to the stimuli presented had a difficult time with these tasks, as illustrated by the poor performance of Algorithms 6 and 7 shown in Figure 21. Algorithms 6 and 7 clearly performed much worse than Algorithm 5, which is not a reinforcement-comparison algorithm, and much worse than Algorithms 8 and 9, which are more sophisticated reinforcement-comparison algorithms.

The performance of the binary-reinforcement Algorithms 1, 2, and 3 on Tasks 1 and 3 was similar to the performance of the corresponding algorithms of Chapter II. Algorithms 2 and 3 performed very poorly on both tasks. Algorithm 1 achieved an intermediate performance level, significantly better than Algorithms 2 and 3, but significantly worse than Algorithms 5, 8, and 9.

Tasks 5 and 6: Reinforcement-Spread Asymmetry. On Tasks 5 and 6 the difference in expected value of reinforcement for the two actions is much larger for one stimulus than it is for the other. For stimulus $\vec{x}^1$, the expected value of reinforcement is .1 higher when the correct action is chosen than when the incorrect action is chosen. For stimulus $\vec{x}^2$, the expected value of reinforcement changes by .3 depending on whether the correct or the incorrect action is selected. These tasks are asymmetrical in the *spread* between their expected reinforcement levels for correct and incorrect actions.

In a task with reinforcement-spread asymmetry it is more important for the learning system to learn the correct action for one stimulus than for the other. A learning algorithm generally learns more rapidly when the reinforcement spread is large than when it is small. Because of this, reinforcement-spread asymmetry tends to cause the correct action for one stimulus (in this case, stimulus $\vec{x}^2$) to be learned much more rapidly than that for the other. When the two stimuli are similar, the more rapid learning to one stimulus generalizes strongly to the other. When two different actions must be learned to the two stimuli, generalization from the more rapidly learning stimulus may overwhelm learning to the other stimulus and cause the incorrect action to become associated with it.

One gets a sense of which algorithms are susceptible to this difficulty by comparing their relative performance rankings on Tasks 5–6 and Tasks 1–2. It is not meaningful to compare the *absolute* performance of algorithms between these pairs

of tasks. On the one hand, Tasks 5–6 might be expected to be more difficult than Tasks 1–2 because of reinforcement-spread asymmetry. On the other hand, Tasks 5–6 might be expected to be easier than Tasks 1–2 because, for one stimulus, the reinforcement spread is larger than in Tasks 1 and 2, which should result in faster learning. Overall it is not possible to predict with certainty which pair of tasks will be the more difficult.

The changes in the relative performance rankings of algorithms from Tasks 1–2 to Tasks 5–6 suggest the following conclusions. First, Algorithm 4 is strongly affected by the reinforcement-spread asymmetry. Whereas on Tasks 1 and 2 it performed the best of all algorithms, on Tasks 5 and 6 it performed significantly worse than Algorithms 5, 8, and 9. Second, all the binary-reinforcement algorithms (1, 2, 3, 10, and 11) perform significantly worse than the better performing algorithms (5, 8, and 9). Algorithm 10 performed significantly better vis-a-vis Algorithms 2, 3, and 11 on Task 5 than it did on Task 1, but its performance was still very poor.

The effect of reinforcement-spread asymmetry on the performance of Algorithms 5, 8, and 9 is less clear. On Tasks 1 and 2 all three algorithms performed equally well. On Task 5, Algorithm 8 performed better than 5 and 9, but only the improvement over Algorithm 5 is statistically significant. On Task 6, Algorithms 5 and 8 both performed significantly better than Algorithm 9.

Tasks 7 and 8: Stimulus-Frequency Asymmetry. On Tasks 7 and 8 one stimulus is presented 3 times as frequently as the other. Learning about the more frequently presented stimulus generally occurs more rapidly than learning about the other, simply because there is more experience with it. This asymmetry in learning rate can lead to an overwhelming of the more slowly forming association by generalization from the faster one, as discussed above for reinforcement-spread asymmetry.

The effect of stimulus-frequency asymmetry on the performances of the various algorithms was very similar to the effect of reinforcement-spread asymmetry. Algorithm 4 was strongly affected, performing significantly worse than Algorithms 5, 8, and 9, whereas it was the best performer on Tasks 1 and 2. On Task 7, the binary-reinforcement algorithms (1, 2, 3, 10, and 11) produced strikingly similar performances to those they produced on Task 5. Algorithm 1 was best, performing at the same level as Algorithm 4. Algorithm 10 was again significantly better than Algorithms 2, 3, and 11, but still performed poorly.

The effect of stimulus-frequency asymmetry on Algorithms 5, 8, and 9 is unclear. Recall that on Tasks 1 and 2 these three algorithms performed equally well. On Task 7 their performances fell in the same rank order as they did on Task 5 (8 then 9 then 5), but here none of the differences are statistically significant. On Task 8, Algorithms 8 and 5 performed better than Algorithm 9, but only Algorithm 8's performance is significantly better.

Tasks 9 and 10: Stimulus-Intensity Asymmetry. On Tasks 9 and 10 one of the two stimuli is 3 times as intense as the other (the two stimuli are given in (11)), whereas in Tasks 1 and 2, the stimuli have the same intensity (see (10)). This is the only difference between Tasks 9 and 10 and Tasks 1 and 2.

Stimulus-intensity asymmetry also causes learning to occur more rapidly for one stimulus than for the other. All learning algorithms considered in these experiments use a multiplicative term ($x_i[t]$) that depends on the intensity of the stimulus presented. Larger changes are thus made in the action-association vector $\vec{w}$ on those steps on which the more-intense stimulus vector occurs than on those steps on which the less-intense stimulus vector occurs. As with stimulus-frequency asymmetry and reinforcement-spread asymmetry, asymmetry in the rate of learning creates the danger of the more slowly forming association being overwhelmed

by generalization from the faster one.

The results on Tasks 9 and 10 were similar to those on Tasks 3–8. Algorithm 4 was again strongly affected by the asymmetry, performing significantly worse than Algorithms 5, 8, and 9, whereas it was the best performer on Tasks 1 and 2. Algorithm 1 performed almost exactly the same as Algorithm 4. The other binary-reinforcement algorithms (2, 3, 10, and 11) performed the worst on these tasks. Although Algorithms 5, 8, and 9 maintained the same ordering (8 then 5 then 9) there are no significant differences between them.

Tasks 11 and 12: Combined Asymmetries. Tasks 11 and 12 combine all four of the asymmetries present individually in Tasks 3–10. Reinforcement-level asymmetry, reinforcement-spread asymmetry, stimulus-frequency asymmetry, and stimulus-intensity asymmetry are combined in such a way that their effects are additive rather than subtractive. As discussed above, each of these asymmetries results in learning to one stimulus proceeding at a more rapid rate than learning to the other stimulus. In Tasks 11 and 12, the favored stimulus with respect to all three asymmetries is $\vec{x}^2$. Stimulus $\vec{x}^2$ is presented with greater frequency, with greater intensity, and has a greater reinforcement spread, than stimulus $\vec{x}^1$.

The experiments involving Tasks 11 and 12 ran for many more steps (see Table 3) than the experiments involving other tasks. One reason for this is that Tasks 11 and 12, since they combine four asymmetries, were expected to be more difficult than Tasks 1–10. However, Tasks 11 and 12 contain weaker forms of them than are present in Tasks 3–10. In addition, the random component of reinforcement for the continuous-reinforcement task (Task 12) has a smaller standard deviation ($\sigma = .025$ instead of $\sigma = .1$), and the reinforcement spreads for the binary-reinforcement task (Task 11) are larger. These differences make both tasks easier.

The real motivation behind the design of Tasks 11 and 12 is to investigate the

convergence behavior of the algorithms. These tasks were designed not to see how rapidly learning proceeds, but to test the ability of various algorithms to ultimately choose the correct actions. This is why the experiments with these tasks were run so many more steps than the others.

The results on Tasks 11 and 12 are not included in the summary figure, but are shown in Figures 19 and 20. The experiments appear to have been run long enough for all algorithms to show the performance levels to which they would have converged, with the exception of Algorithm 2 on Task 11 and Algorithm 5 on Task 12. All the other algorithms appear to converge to one of two performance levels, shown as dashed lines in each graph. The higher of the two performance levels indicated by dashed lines is the maximum possible performance level on these tasks. This performance level is achieved if the correct action is chosen with probability one for both stimuli. Algorithms 1, 5, 8, and 9 appear to converge to this performance level on Task 11, and Algorithms 8 and 9 appear to converge to this performance level on Task 12. Task 12 appears not to have been run long enough for Algorithm 5 to converge. The highest performance level for Algorithm 5 was significantly lower than that of Algorithm 8 and 9. Most likely, this Algorithm also converges to the optimal performance level on this task, but converges more slowly than Algorithms 8 and 9.

The lower of the two dashed lines shown in each graph indicates the performance level achieved if action 0 is selected with probability one in response to both stimuli. This action is the better action in response to one stimulus but the poorer action in response to the other. Convergence to this performance level thus indicates a failure to discriminate between the two stimuli. The performances of Algorithms 3, 4, 10, and 11 appear to converge to this level on Task 11, and Algorithm 4 also appears to converge to this level on Task 12.

These results on Tasks 11 and 12 do not prove convergence or lack of con-

vergence for any algorithm. Algorithms that appear to converge to one of the performance levels indicated may actually converge to slightly different levels. Algorithms that fail to converge during these experiments, or that appear to converge to a suboptimal level, may in fact converge to the optimal performance level if sufficiently small α values are used and the experiments are run long enough. The only way to definitively prove such convergence is by mathematical analysis. These results do, however, strongly suggest what the results of such an analysis might be and provide practical indications of how the algorithms are likely to perform.

General Discussion

There are a number of similarities between the results of this chapter's experiments and those of Chapter II. Here, as there, the algorithms designed purely for binary-reinforcement tasks (Algorithms 1, 2, 3, 10, and 11), with the exception of Algorithm 1, performed uniformly poorly across tasks. In the experiments of both chapters, Algorithm 1 performed better than the other binary-reinforcement algorithms, but performed clearly worse than Algorithms 5, 8, and 9. In both chapters, Algorithm 4 varied in its performance across tasks, performing best on a few, poorly on a few, and significantly worse than the best algorithms on the others.

As anticipated, the algorithms that simply compare current and immediately preceding reinforcement levels (Algorithms 6 and 7) performed comparatively much worse on these associative-learning tasks than on the nonassociative tasks of Chapter II. These algorithms performed much worse than Algorithms 5, 8, and 9 on the tasks of this chapter.

The best performing algorithms across tasks in this chapter were Algorithms 5, 8, and 9. It is not entirely clear which of these was best. Of the three, Algorithm

9 performed significantly better on Tasks 3 and 4, but also performed significantly worse than Algorithm 8 on Tasks 8 and 10, and significantly worse than both Algorithms 5 and 8 on Task 6. The only completely clear ordering that can be made among these three algorithms is that Algorithm 8 performed better than Algorithm 5. Algorithm 8 performed as well or better than Algorithm 5 on every task, and significantly better on Tasks 3, 4, 5, and 10. In addition, both Algorithm 8 and Algorithm 9 performed significantly better than Algorithm 5 on Tasks 11 and 12.

Whether Algorithm 8 or Algorithm 9 is the better algorithm apparently depends on the nature of the task. Although Algorithm 8 out-performed Algorithm 9 on all tasks except Tasks 3 and 4, it is possible that "real life" learning tasks resemble Tasks 3 and 4 more closely that any of the other tasks considered. Each of these tasks is a special case, and no attempt has been made to characterize what a real "typical case" might look like. However, the fact that on these tasks Algorithm 8 makes a serious bid for the title of best performer is significant in light of the results of Chapter II, in which Algorithm 8 was strictly inferior or equal to Algorithm 9. Somehow the change from nonassociative to associative tasks makes Algorithm 8 a better performer vis-a-vis Algorithm 9.

Since Algorithm 5 is strictly inferior to Algorithm 8 in its performance on this chapter's tasks, these results corroborate the results of Chapter II with regard to reinforcement-comparison algorithms; in the experiments of both chapters the best performing algorithms across tasks are reinforcement-comparison algorithms.

As discussed earlier, and as is illustrated by the poor performance of Algorithms 6 and 7, construction of a satisfactory reinforcement-comparison algorithm for associative learning is not a trivial task. The difficulties may be part of the reason researchers have avoided reinforcement-comparison algorithms, even for some types of nonassociative learning. In this regard, the excellent performance of the more

sophisticated reinforcement-comparison algorithms, Algorithms 8 and 9, is particularly important. It demonstrates that algorithms can be designed for associative learning that can benefit from the advantages of a reinforcement-comparison mechanism. The advantages of reinforcement comparison can be obtained in associative-learning tasks as well as in nonassociative-learning tasks.

Perhaps the most important result of the experiments described in this chapter is the poor performance of Algorithm 4. Not only does Algorithm 4 perform much worse than Algorithms 5, 8, and 9 on all tasks involving any form of stimulus asymmetry, but the results of Tasks 11 and 12 suggest that this algorithm is unable to discriminate stimuli properly. On Tasks 11 and 12 this algorithm appears to converge to the incorrect choice of action and remain there. Yet Algorithm 4 appears to be the most straightforward implementation of the basic principle of associative reinforcement learning using the linear-mapping approach. In fact, of the algorithms considered here, Algorithm 4 is the algorithm most nearly like that of Farley and Clark (1954), which is one of the few associative reinforcement learning algorithms using the linear-mapping approach that has been computationally investigated. This also appears to be essentially the algorithm informally discussed by Minsky and Selfridge (1961). The failure of this common-sense rule to produce effective learning in all but the simplest situations suggests that associative reinforcement learning involves subleties that were not recognized by early AI and cybernetic researchers.

CHAPTER IV

DELAYED REINFORCEMENT

The experiments of preceding chapters concern learning tasks in which actions affect only the reinforcement received on the following time step. Henceforth such tasks are referred to as ***immediate-reinforcement*** tasks. Tasks in which some of the effects of an action on reinforcement are delayed by two or more time steps are called ***delayed-reinforcement*** tasks. In a delayed-reinforcement task, r[t+1] may depend on any of $y[t]$, $y[t-1]$, $y[t-2]$, ...

Our interest in delayed reinforcement stems from the fact that difficult temporal credit-assignment problems are almost always partially due to delayed reinforcement. A major reason for the difficulty in assigning credit for the outcome of a chess game, for example, is the potentially great delay between critical moves and the outcome. Any task in which reinforcement is influenced by sequences of actions involves delayed reinforcement. Further, the whole point of secondary reinforcement mechanisms (see Chapter I) is to reduce delays between actions and reinforcement.

Delayed reinforcement is important enough as an isolated topic with respect to the overall aims of this dissertation to justify a chapter devoted to it. Accordingly, the tasks investigated in this chapter are such that they offer no opportunities for secondary reinforcement. Instead, this chapter deals with more elementary issues: the effect of delay on learning rate, and the extension of the results and algorithms

of preceding chapters to the case of delayed reinforcement.

It is the ***uncertainty*** that usually accompanies delayed reinforcement, rather than the delay itself, that creates a difficult temporal credit-assignment problem. If reinforcing events are always delayed by a fixed amount known ***a priori***, then the design of the learning system can take the delay into account, and credit assignment need not be more difficult than it is with immediate reinforcement. On the other hand, if the length of delay is not known ***a priori***, then uncertainty about the causal relationship between action and reinforcement is created, making temporal credit assignment genuinely more difficult.

The recency and frequency heuristics studied in this chapter are conventionally thought of as "general but weak" methods, as opposed to the knowledge-rich "specific but powerful" methods that are central to much of modern AI. The intention in studying these heuristics has not been to dispel this conventional assessment, but to establish and detail it experimentally. The results reported here are not surprising ones; they are roughly what one might expect. Their importance is primarily in confirming and demonstrating what would otherwise be presumption, and in detailing the magnitude and generality of the various phenomena.

Eligibility Traces

One cannot expect to find algorithms that completely eliminate the temporal credit-assignment problems caused by delayed reinforcement. One can, however, attempt to find learning algorithms whose performance degrades gracefully as uncertainty due to delay between action and reinforcement is increased. As is shown below, the algorithms considered in previous chapters do not learn effectively on delayed-reinforcement tasks. How can they be modified to improve their performance on such tasks?

Assume for the moment that there is available a heuristic reinforcement signal $\hat{r}$ that already incorporates reinforcement-comparison mechanisms, but whose evaluations are delayed. Assume that a complete record of all past actions and stimuli is also available. How is credit for the reinforcement received at the current time to be allocated to past behavior? Clearly there are many heuristics that could be used to assign credit differentially to some of that behavior more than the rest. Here we consider two of the most general heuristics for assigning credit in the face of delayed reinforcement, those of *frequency* and *recency*.

According to the *frequency heuristic*, one assigns credit to past decisions according to how many times they occurred. If one action had been made once in response to a particular stimulus in the time preceding reinforcement, and another action had been made twice, then the second action, according to the frequency heuristic, is twice as likely to have caused the reinforcement and thus deserves twice as much credit for it. According to the *recency heuristic*, one assigns credit for current reinforcement to past actions according to how recently they were made. Credit assigned should be a monotonically decreasing function of the time between action and reinforcement, approaching zero as this time approaches infinity. One way of doing this is to assign credit according to an exponentially decreasing function λ^k of the number k of time steps elapsing between action and reinforcement.

The following learning rule combines the frequency heuristic and an exponentially decaying recency heuristic:

$$w_i[t+1] = w_i[t] + \alpha\,\hat{r}[t+1](1-\lambda)\sum_{k=0}^{t-1}\lambda^k e_i[t-k], \tag{14}$$

for $t > 0$ and $i = 1, \ldots, n$, where $\vec{w}[t] = (w_1[t], \ldots, w_n[t])$ is the vector of action-association weights (mapping stimuli to actions), $e_i[t]$ is the eligibility of its ith component (as discussed in Chapter III), α is a positive learning-rate parameter, and $\hat{r}[t+1]$ is heuristic reinforcement. The eligibility $e_i[t]$ can be regarded as the credit that should be assigned to w_i for a "unit credit" assigned to the behavior

at time t. λ^k is the credit that should be assigned, according to an exponential recency heuristic, to the behavior at time $t-k$ given that a unit reinforcement is received at time $t+1$. The product $\hat{r}[t+1]\lambda^k e_i[t-k]$ therefore is the credit due to the behavior at $t-k$ that should be assigned to w_i given $\hat{r}[t+1]$. The frequency heuristic is implemented in this rule by summing up the credit due to the behavior at all past times. The term $(1-\lambda)$ in the learning rule normalizes the sum. If λ is near 1, then the credit assigned to an action decreases slowly as the time between it and reinforcement increases; if λ is near 0, then credit decreases rapidly. For $\lambda = 0$, (14) reduces to the form used in the learning rules of Chapter III. Henceforth, $\delta = 1 - \lambda$.

Equation 14 implements frequency and recency heuristics by means of an exponentially decaying backwards-averaging kernel. It is reasonable to regard a kernel of this general form as appropriate for learning situations in which nothing specific is known about underlying cause and effect relationships. Alternatively, it might be justifiable to assign credit according to an inverted-U shaped function of the time between action and reinforcement as suggested by Klopf (1972, 1982). This choice could be regarded as reflecting the distribution of the durations of the feedback pathways in which the learning system is embedded. In general, one would expect learning performance to improve to the extent that the shape of this kernel incorporates task-specific knowledge about the temporal relationship between cause and effect. Ideally, perhaps, the kernel should resemble the cross-correlation of the action and reinforcement processes. In the research reported here, no attempt has been made to use any knowledge of this type, even though *some* amount of such knowledge is likely to be available about specific environments. Nor have algorithms been considered for adaptively adjusting the form of the kernel (e.g., by adjusting λ, or by estimating the action/reinforcement cross-correlation), but such methods have obvious utility.

A major advantage of the exponential decay form of the recency heuristic is

that it allows the sum in (14) to be computed iteratively. For any time sequence e_i, let $\bar{e}_i$ denote the time sequence resulting from the application of a ***trace operator*** to e_i, defined as follows:

$$\bar{e}_i[t] = (1-\lambda)\sum_{k=0}^{t-1}\lambda^k e_i[t-k],$$

for $t > 0$, $\bar{e}_i[0] = 0$. An algorithm for iteratively computing this trace can be derived as follows:

$$\begin{aligned}\bar{e}_i[t] &= (1-\lambda)\sum_{k=0}^{t-1}\lambda^k e_i[t-k] \\ &= (1-\lambda)e_i[t] + (1-\lambda)\sum_{k=0}^{t-2}\lambda^{k+1}e_i[t-1-k] \\ &= (1-\lambda)e_i[t] + \lambda\bar{e}_i[t-1],\end{aligned} \tag{15}$$

for $t > 0$. Although it requires only a single memory variable per component of $\vec{w}$, the above iterative algorithm is equivalent to remembering all past behavior and then applying frequency and exponential recency heuristics as in (14). The algorithm given by (15) is a standard discrete-time recursive linear filter.

When e_i is an eligibility time sequence as defined in Chapter III, we call $\bar{e}_i$ an ***eligibility trace.*** All algorithms of this chapter update $\vec{w}$ by (14), which can be rewritten using $\bar{e}_i$ as:

$$w_i[t+1] = w_i[t] + \alpha\,\hat{r}[t+1]\bar{e}_i[t]. \tag{16}$$

When $\lambda = 0$ ($\delta = 1$), $\bar{e}_i$ reduces to e_i, and (16) reduces to the same form as used in the algorithms of Chapter III. The term eligibility will be used to refer to $\bar{e}_i$ when it is used as in (16). The even-numbered algorithms of this chapter (4, 8, and 10) define $e_i[t]$ as $(y[t]-\frac{1}{2})x_i[t]$, and the odd-numbered algorithms (5, 9, and 11) define $e_i[t]$ as $(y[t]-\pi[t])x_i[t]$, where $y[t] \in \{0,1\}$ is the action taken at time t, $\pi[t]$ is the probability that $y[t] = 1$, and $x_i[t]$ is the ith component

of the stimulus vector at time t. Otherwise, the algorithms differ only in the reinforcement-comparison mechanism used in forming $\hat{r}$, the subject considered next. Table 6 summarizes all of this chapter's algorithms.

The recency and frequency heuristics embodied in eligibility traces make only weak assumptions about the length and nature of the delay between action and reinforcement. If more information is known *a priori* or can be learned, it could in many cases be built into a more sophisticated eligibility-trace mechanism. Such methods could contribute to easing the problems introduced by delayed reinforcement, but are beyond the scope of this dissertation. It is likely that such enhancements would be compatible with the heuristic-reinforcement mechanisms that are the primary focus in this dissertation.

Reinforcement Comparison Under Delayed Reinforcement

The reinforcement-comparison algorithms of preceding chapters can be viewed as constructing a reinforcement signal $\hat{r}$ by comparing the primary reinforcement r with a ***predicted reinforcement*** $p[t]$:

$$\hat{r}[t+1] = r[t+1] - p[t]. \tag{17}$$

Any reinforcement signal such as this is called a ***heuristic reinforcement*** signal. As discussed in Chapter III, the predicted reinforcement $p[t]$ is computed from the stimulus vector $\vec{x}[t]$ by means of a ***reinforcement-association vector*** $\vec{v}[t]$:

$$p[t] = \sum_{i=1}^{n} v_i[t] x_i[t]. \tag{18}$$

The reinforcement-association vector $\vec{v}$ is updated according to a variation of the

Table 6. Learning Algorithms of Chapter IV.

Algorithm	Update Rule
4,5	$w_i[t+1] = w_i[t] + \alpha r[t+1]\bar{e}_i[t]$
8,9	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - p[t])\bar{e}_i[t]$
	$v_i[t+1] = v_i[t] + \beta(r[t+1] - p[t])\bar{x}_i[t]$
10,11	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - \bar{p}[t])\bar{e}_i[t]$
	$v_i[t+1] = v_i[t] + \beta(r[t+1] - \bar{p}[t])\bar{x}_i[t]$

Where:

$$e_i[t] = \begin{cases} (y[t] - \frac{1}{2})x_i[t], & \text{for even numbered algorithms;} \\ (y[t] - \pi[t])x_i[t], & \text{for odd numbered algorithms,} \end{cases}$$

$$w_i[0] = 0, \qquad v_i[0] = 0, \qquad y[t] \in \{1, 0\}, \qquad \alpha > 0, \qquad \beta = .05, \qquad 0 < \gamma < 1,$$

and $\pi[t]$ is the probability that y[t]=1, given $\vec{x}[t]$.

$$y[t] = \begin{cases} 1, & \text{if } s[t] + \eta[t] > 0; \\ 0, & \text{otherwise,} \end{cases}$$

where $\eta[t]$ is a normally distributed random variable of mean 0 and standard deviation .1, and

$$s[t] = \sum_{i=1}^{n} w_i[t]x_i[t].$$

$$p[t] = \sum_{i=1}^{n} v_i[t]x_i[t].$$

For any time sequence, e.g., $z[t]$, $\bar{z}[t]$ is defined by

$$\bar{z}[t] = (1-\delta)\bar{z}[t-1] + \delta z[t], \qquad \bar{z}[0] = 0, \qquad 0 < \delta < 1.$$

Widrow-Hoff rule (Widrow and Hoff, 1960):

$$v_i[t+1] = v_i[t] + \beta\,\hat{r}[t+1]x_i[t], \tag{19}$$

where $v_i[0] = 0$, and β is a positive constant. Since $p[t]$ is an estimate of $r[t+1]$, $\hat{r}[t+1] = r[t+1] - p[t]$ is an error term. This learning rule correlates these errors with the stimuli that were present immediately before them and adjusts $\vec{v}$ in such a way as to reduce the error.

One way to generalize (19) to delayed reinforcement is to correlate the error $\hat{r}$ with all preceding stimuli, weighted according to their frequency and their recency in a manner similar to that discussed above for weighting past eligibilities:

$$\begin{aligned} v_i[t+1] &= v_i[t] + \beta\,\hat{r}[t+1](1-\lambda)\sum_{k=0}^{t-1}\lambda^k x_i[t-k] \\ &= v_i[t] + \beta\,\hat{r}[t+1]\bar{x}_i[t+1], \end{aligned} \tag{20}$$

where λ, $0 < \lambda < 1$, is an exponential decay rate and $\bar{x}_i[t]$ is due to $x_i[t]$ via the trace operator discussed above.

Equations (17), (18) and (20) constitute the mechanism used by Algorithms 8 and 9 of this chapter for constructing and updating their heuristic reinforcement signal. Algorithms 4 and 5, on the other hand, use $\hat{r}[t] = r[t]$ and so do not require the computation of predicted reinforcement.

The reinforcement-comparison mechanism of Algorithms 10 and 11 is based on the idea that, on delayed reinforcement tasks, the prediction of reinforcement at time $t+1$ should depend on all past stimuli rather than on just the stimulus at time t. These algorithms compare primary reinforcement with a trace of the past predicted reinforcement levels:

$$\hat{r}[t+1] = r[t+1] - \bar{p}[t]. \tag{21}$$

Algorithm 10 uses $\hat{r}$ as given by (21) both to update the action-association vector $\vec{w}$ by (16) and the reinforcement-association vector $\vec{v}$ by (19).

Experiment 1: The Effect of Delay

In this experiment, 7 nonassociative tasks of varying delays are used to illustrate how severely delayed reinforcement can influence the learning process. Each task is a continuous-reinforcement task with a balanced reinforcement distribution. When the correct action (Action 1) is taken, the expected value of reinforcement is $+.1$, otherwise it is $-.1$. In either case, the reinforcement is chosen from a normal distribution of standard deviation $\sigma_r = .1$.

The 7 tasks differ only in the length of the delay between the time of an action and the time of delivery of the resultant reinforcement. Delays of 0, 5, 10, 15, 20, 25, and 30 time steps were used, where a delay of 0 means that $r[t+1]$ is due to $y[t]$, as in the tasks of preceding chapters, and a delay of 5 means that $r[t+1]$ is due to $y[t-5]$, etc.

A major purpose of this experiment is to compare the difficulty of tasks as a function of the length of the delay between action and reinforcement. Therefore, care must be taken regarding the number of steps of each run. If each run ran 15 steps, then runs with a delay of 0 would include 15 reinforcements relevant to the actions taken on the run, whereas runs with nonzero delays would include fewer. Runs with delays of 15 or more would include no reinforcements relevant to the actions taken. To ensure that the same number of relevant reinforcements were delivered regardless of the delay, each run was extended beyond 15 steps by the length of the delay. The runs with zero delay ran 15 steps, those with a delay of 5 ran 20 steps, etc. For those runs with a nonzero delay, the reinforcement for some of the first steps (as many as the length of the delay) were not influenced by any

action, and were zero.

Only Algorithms 4 and 5 were used in Experiment 1. As discussed above, these two algorithms are straightforward extensions of the like-numbered algorithms of Chapter III. They include exponentially decaying eligibility traces. If δ were chosen to be 1, then Algorithms 4 and 5 of the two chapters would be identical. For each task, i.e., for each delay value, and for each of these two algorithms, 200 runs were simulated for each combination of a range of values for the α and δ parameters. The α values used were the powers of two from 2^{-1} to 2^{13}, and the δ values used were the powers of 2 from 2^0 to 2^{-6}.

The algorithms listed in Table 6 are all written as if they were associative-learning algorithms in that they include an x_i factor dependent on the stimulus in their eligibility factors. In the experiments of this chapter that involve nonassociative tasks, all vectors ($\vec{w}$, $\vec{v}$, and $\vec{x}$) were taken instead to be scalars, with the stimulus $x[t]$ always equal to 1. For example, with these changes, the nonassociative form of the update rule (16) is

$$w[t+1] = w[t] + \alpha\, \hat{r}[t+1]\bar{e}[t]$$

with

$$e[t] = \begin{cases} y[t+1] - \frac{1}{2}, & \text{for even-numbered algorithms;} \\ y[t+1] - \pi[t], & \text{otherwise.} \end{cases}$$

At the end of each run the probability of a correct choice on the next step was computed. The average of this probability over 200 runs is the measure of performance plotted in the graphs and discussed below.

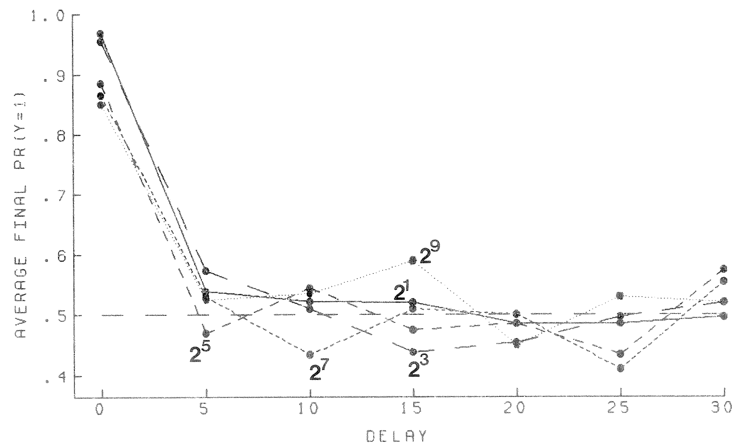
By looking selectively at different parts of the large array of data generated by this experiment, various issues regarding the effect of delayed reinforcement on learning can be highlighted. The first issue is how well the algorithms without eligibility traces perform on delayed-reinforcement tasks. Recall that when $\delta = 1$ there are no traces; the algorithms are the same as those of the preceding chapters.

Figure 22 plots performance versus delay for Algorithms 4 and 5 with $\delta = 1$. In both cases the data are shown for a few representative values for α ($\alpha = 2^1, 2^3, 2^5, 2^7$, and 2^9). The data for Algorithm 4 are in the upper graph and the data for Algorithm 5 are in the lower graph. Note how rapidly performance falls as delay increases. For any nonzero delay these traceless algorithms performed only at, or very near, chance level.

With exponentially decaying eligibility traces, the performances of Algorithms 4 and 5 degrade more slowly as delay length increases. Figures 23, 24, and 25 present the same data as in Figure 22, but from simulation runs with lower values for δ, resulting in longer traces. While Figure 22 presents the performance of Algorithms 4 and 5 for 5 values of α with $\delta = 1$, Figures 23, 24, and 25 plot the corresponding performances for $\delta = 2^{-1}$, 2^{-4}, and 2^{-6} respectively. Note that even with traces, the performances of these algorithms drop sharply as the delay between action and reinforcement increases. Traces can make learning possible with delayed reinforcement, but they cannot prevent it from becoming much slower.

Also note that although traces increase performance on the tasks with delayed reinforcement, they tended to decrease performance on the immediate reinforcement task. This trade-off is seen most clearly in Figure 26 which shows performance as a function of delay for several different values of δ. These graphs are all due to a single value of α, $\alpha = 2^3$. Each curve shows the performance of the algorithm, at this α value, for the particular value of δ that is marked near the curve. The use of longer traces (smaller δ's) enabled considerable learning to take place even at long delays. Note that the low-δ versions of each algorithm performed best when reinforcement was delayed, and the high-δ versions performed best when reinforcement was immediate.

If the delay length was known *a priori*, or could be learned, then this trade-off could be avoided. The highest performance levels, obtained for the immediate-



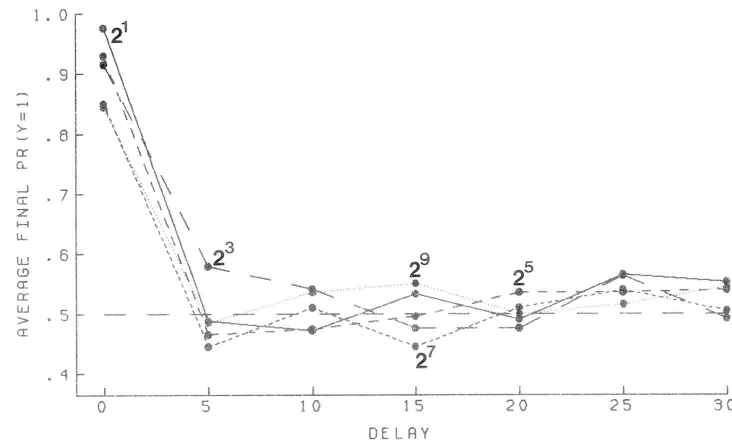
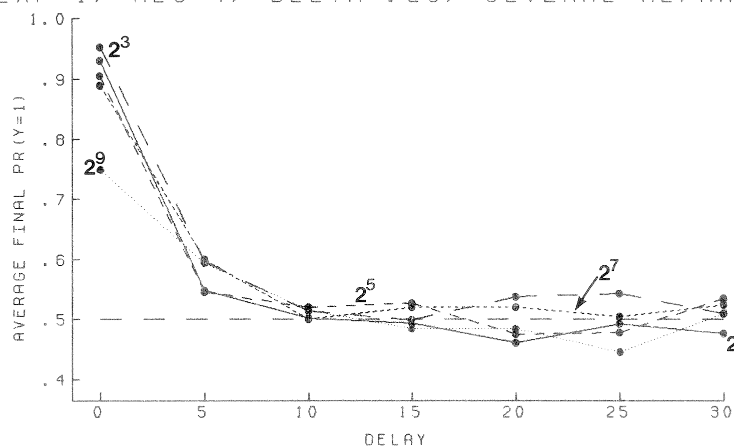


Figure 22. Effect of Delayed Reinforcement on the Performance of Algorithms Without Eligibility Traces. Each point represents the average performance of the algorithm over all runs on a particular task with a particular α value, where performance on a run is defined as the probability of selecting the correct action at the end of the run. Points due to runs with the same α value (on different tasks) are connected by lines; the label indicates the associated α value. The tasks differ from each other in the length of the delay between action and resultant reinforcement; the delay lengths are given along the x-axis in time steps. The horizontal dashed line indicates the initial, chance performance level.



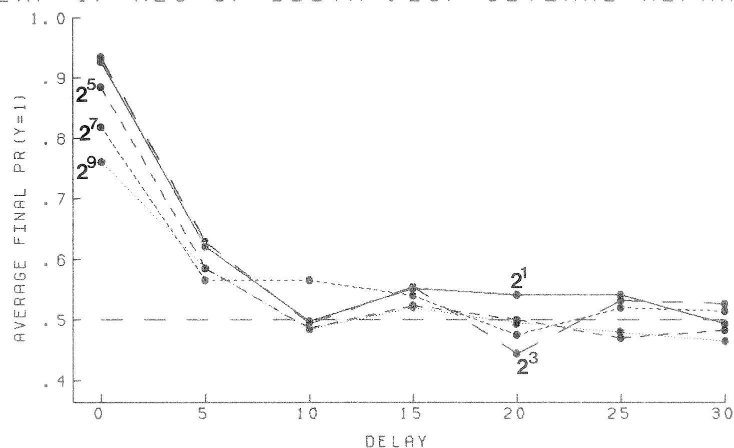
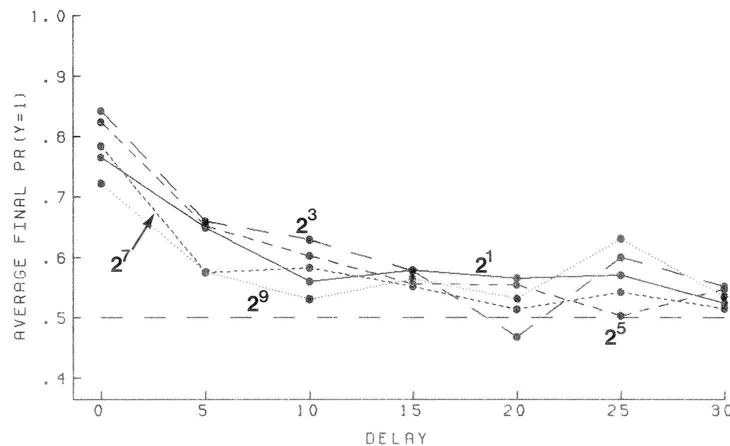


Figure 23. Effect of Delayed Reinforcement on the Performance of Algorithms 4 and 5 with Short Eligibility Traces. Each point represents the average performance of the algorithm over all runs on a particular task with a particular α value, where performance on a run is defined as the probability of selecting the correct action at the end of the run. Points due to runs with the same α value (on different tasks) are connected by lines; the label indicates the associated α value. The tasks differ from each other in the length of the delay between action and resultant reinforcement; the delay lengths are given along the x-axis in time steps. The horizontal dashed line indicates the initial, chance performance level.



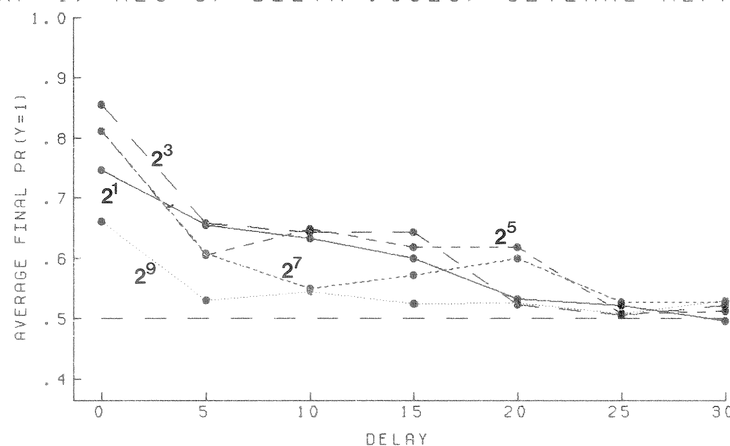
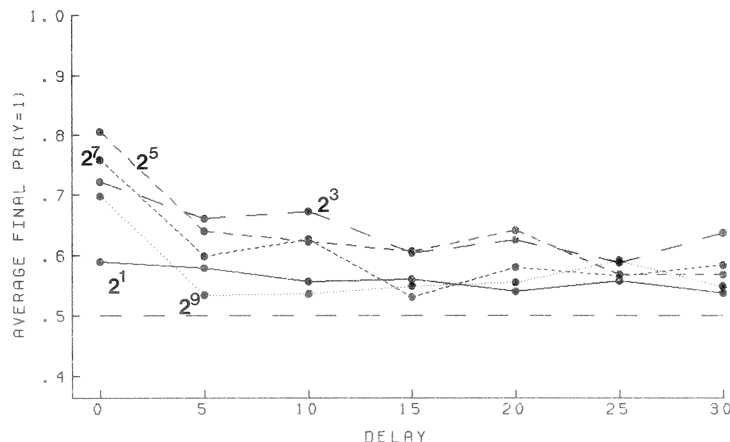


Figure 24. Effect of Delayed Reinforcement on the Performance of Algorithms 4 and 5 with Moderate Length Eligibility Traces. Each point represents the average performance of the algorithm over all runs on a particular task with a particular α value, where performance on a run is defined as the probability of selecting the correct action at the end of the run. Points due to runs with the same α value (on different tasks) are connected by lines; the label indicates the associated α value. The tasks differ from each other in the length of the delay between action and resultant reinforcement; the delay lengths are given along the x-axis in time steps. The horizontal dashed line indicates the initial, chance performance level.



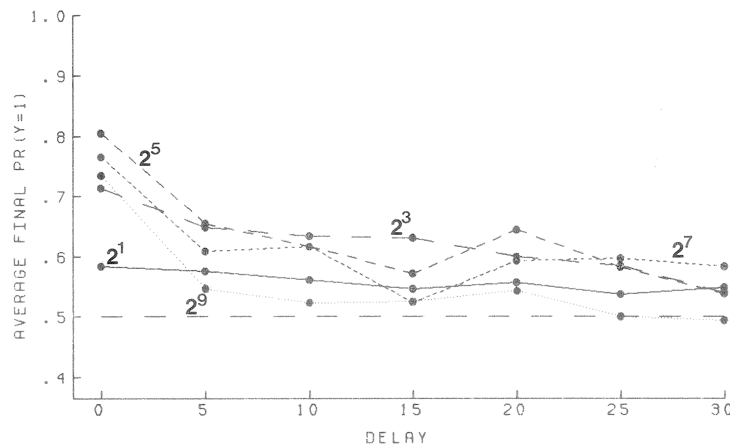


Figure 25. Effect of Delayed Reinforcement on Performance of Algorithms with Long Eligibility Traces. Each point represents the average performance of the algorithm over all runs on a particular task with a particular α value, where performance on a run is defined as the probability of selecting the correct action at the end of the run. Points due to runs with the same α value (on different tasks) are connected by lines; the label indicates the associated α value. The tasks differ from each other in the length of the delay between action and resultant reinforcement; the delay lengths are given along the x-axis in time steps. The horizontal dashed line indicates the initial, chance performance level.

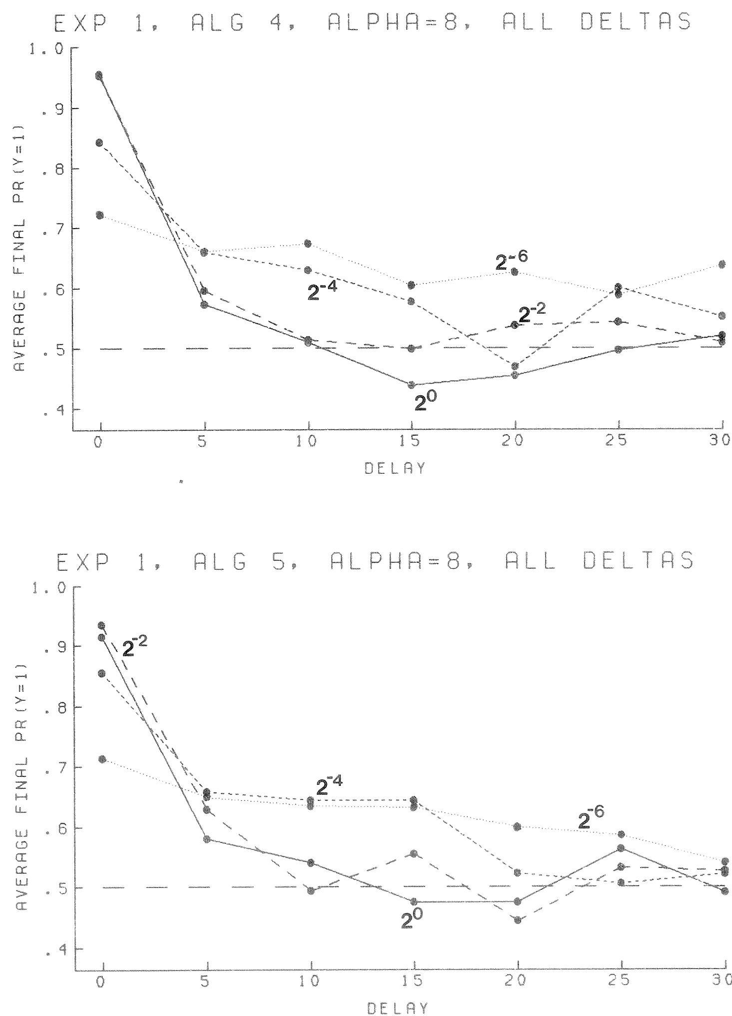


Figure 26. Interaction of Eligibility Trace Length and Performance on Delayed-Reinforcement Tasks. Each point represents the average performance of the algorithm over all runs on a particular task with a particular eligibility trace length, where performance on a run is defined as the probability of selecting the correct action at the end of the run. $\alpha = 8$ for all runs whose data is shown here. Points due to runs with the same trace length (on different tasks) are connected by lines; the label indicates the associated value of the trace decay parameter δ. The tasks differ from each other in the length of the delay between action and resultant reinforcement; the delay lengths are given along the x-axis in time steps. The horizontal dashed line indicates the initial, chance performance level.

reinforcement task without traces, and shown in Figure 22, could then ideally be obtained for all delay lengths.

Experiment 2: Sooner Is Better

According to the recency heuristic, the sooner a reinforcing event follows an action the larger its effect in encouraging the action to recur. This heuristic involves a danger of weighting quick reinforcement more heavily than is appropriate. Experiment 2 was designed to demonstrate this danger for the recency heuristic as embodied in eligibility traces.

Experiment 2 involved a single continuous-reinforcement task in which primary reinforcement is defined by

$$r[t+1] = .1\, y[t] + .2\,(1 - y[t-2]),$$

for $t > 2$, where $y[t] \in \{0,1\}$ is the action selected at time t. Both Actions 1 and 0 have positive influences on subsequent reinforcement, but whereas Action 0 has a $+.1$ influence on the immediately following reinforcement, Action 1 has a $+.2$ influence on the reinforcement 2 steps later. If these two influences coincide, then they add, and the resultant reinforcement is $+.3$. If neither influence applies to a reinforcement value, then it is 0. There is no random component to the reinforcement signal on this task.

If Action 1 were selected every time step, a reinforcement of $+.2$ would occur every time step, and if Action 0 were selected every step, a reinforcement of $+.1$ would occur every step. Action 1 is clearly the correct action in the long run, but in the initial stages of learning when Actions 1 and 0 are both being tried intermittently, Action 0 has some advantage because the reinforcement it causes is delivered more quickly than that caused by Action 1.

Does this asymmetry regarding the speed with which reinforcement is delivered seriously disrupt either the speed of learning or the ability to select the better action for some algorithms? To measure the extent of disruption, performance on this task must be compared with that on a task without the asymmetry. One reasonable such ***control task*** is that with the same reinforcement levels for the two actions, $+.1$ and $+.2$, but with delays both equal to the length of the longer delay in the asymmetrical task (2 time steps). This control task differs from the asymmetrical one only in that one delay is lengthened from 0 to 2 steps. The results of Experiment 1 show that lengthening the delay under these conditions normally makes learning more difficult. Therefore, if an algorithm performs better on the control task than on the asymmetrical task, then it must be due to the removal of a harmful effect of the asymmetry.

For both the control task and the asymmetrical task, simulations were run for all 6 algorithms with each combination of a range of values for the parameters α and δ. The α values used were the powers of 2 from 2^{-5} to 2^{3}. The δ values used were the powers of two from 2^{0} to 2^{-6}. For each task, algorithm, and parameter setting, 500 runs of exactly 200 steps were simulated. At the end of the 200 steps of each run, the probability $\pi[201]$ of performing the correct action (Action 1) on the next step was recorded. This probability was averaged over the 500 runs to yield a measure of performance on each task for each algorithm at each parameter setting.

Figure 27 plots the highest average performance level achieved with any of the values of α tried for each algorithm, δ value, and for both the asymmetrical and control tasks. The upper group of plots are due to performances on the control task, and the lower group of plots are due to performances on the asymmetrical task. The horizontal dashed line indicates the performance level attained if actions are selected totally at random. This is also the initial performance level.

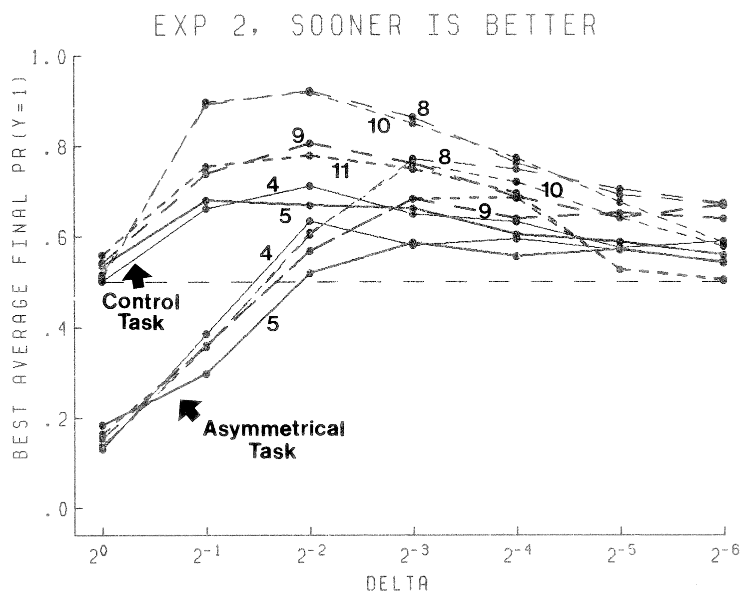


Figure 27. Comparison of Algorithm Performance with and without Delay Asymmetry. The higher group of data points are performances on the control task, and the lower group are performances on the asymmetrical task. Each point represents the average performance level of a particular algorithm with a particular eligibility trace length (δ value), and with the α value at which performance was best for that algorithm. Points due to the same algorithm (with different δ values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the chance performance level. Note that all algorithms performed worse on the asymmetrical task than they did on the control task.

All algorithms performed significantly worse on the asymmetrical task than they did on the control task. The best performance of every algorithm on the control task is significantly better ($P < .01$) than its best performance on the experimental task. In addition, at each of the first four δ values, where each algorithm achieved its best performance, every algorithm performed significantly better ($P < .01$) on the control task than on the asymmetrical task. This strik-

ingly poor performance by all algorithms on the asymmetrical task illustrates how debilitating this asymmetry can be.

For $\delta = 1$ and $\delta = \frac{1}{2}$ (the two highest δ values), every algorithm performed significantly worse ($P < .01$) than the chance level. This indicates that algorithms performed poorly on the asymmetrical task because they actually learned to choose the wrong action. In these cases better performance would have been attained with a learning-rate parameter of $\alpha = 0$.

It is possible to mathematically determine the "critical value" for δ, above which the incorrect action is learned, and below which the correct action is learned. As discussed earlier, the eligibility of an action decays as $\lambda^k = (1-\delta)^k$, where k is the number of time steps since the action. Since the effect of reinforcement on an action is proportional to both the size of the reinforcement and the eligibility of the action, the effect of a reinforcement of size r delayed by k time steps is $r(1-\delta)^k$. Whether a quick, small reinforcement, or a slow, large reinforcement is the more effective depends on which results in a larger value for $r(1-\delta)^k$. The critical value of δ, therefore, is that at which $r(1-\delta)^k$ is equal for the two reinforcements. For the reinforcement sizes and delays in Experiment 2, this value of δ is that at which:

$$.2(1-\delta)^2 = .1(1-\delta)^0.$$

Solving for δ yields $\delta = 1 - \sqrt{1/2} \approx .293$. Since the first two δ values used in Experiment 2 are above this value, and these are the cases in which the incorrect action was learned, this value is consistent with the simulation results.

The primary result of this experiment is that *all* algorithms performed poorly in the face of different delay lengths for different actions. This experiment also indicates some statistically significant differences between algorithms. In particular, the reinforcement-comparison algorithms and the algorithms using eligibility factors involving $y[t] - \frac{1}{2}$ performed better than those without these characteristics.

Experiment 3: Comparison of Algorithms on Nonassociative Tasks

Experiment 3 was essentially a repetition of the three continuous-reinforcement, nonassociative tasks examined in Chapter II, but with delayed reinforcement. The intent, as in Chapter II, was to compare the performance of the various algorithms.

Experiment 3 involved 3 tasks. On all 3 tasks reinforcement is delayed by 5 time steps for both actions. On all tasks Action 1 is the better action. The tasks are called "high," "low," and "middle" tasks, according to their distribution of reinforcement values; The high and low tasks are unbalanced-reinforcement tasks, and the middle task is a balanced-reinforcement task. On the high task, selection of Action 1 results in reinforcement 5 steps later being $+.2$, whereas selection of Action 0 results in it being $+.1$. The corresponding reinforcement values for the low task are $-.1$ and $-.2$, and for the middle task, $+.05$ and $-.05$, for Actions 1 and 0 respectively. On all tasks reinforcement depends deterministically on the action chosen.

For all three tasks, simulation runs were performed with all 6 algorithms with each combination of a range of values for the parameters α and δ. The α values used were the powers of 2 from 2^{-3} to 2^9. The δ values used were the powers of 2 from 2^0 to 2^{-7}. For each task, algorithm, and parameter setting, 200 runs of exactly 200 steps each were simulated. At the end of the 200 steps, the probability $\pi[201]$ of performing the correct action (Action 1) on the next step was computed. This probability, averaged over the 200 runs, is the performance measure used for each combination of algorithm and parameter setting on each task.

Figures 28, 29, and 30 each summarize the data from one task of Experiment 3. Each point in these graphs represents the best performance level of a particular algorithm at a particular δ value on a particular task, i.e., the performance for the α value at which performance was best.

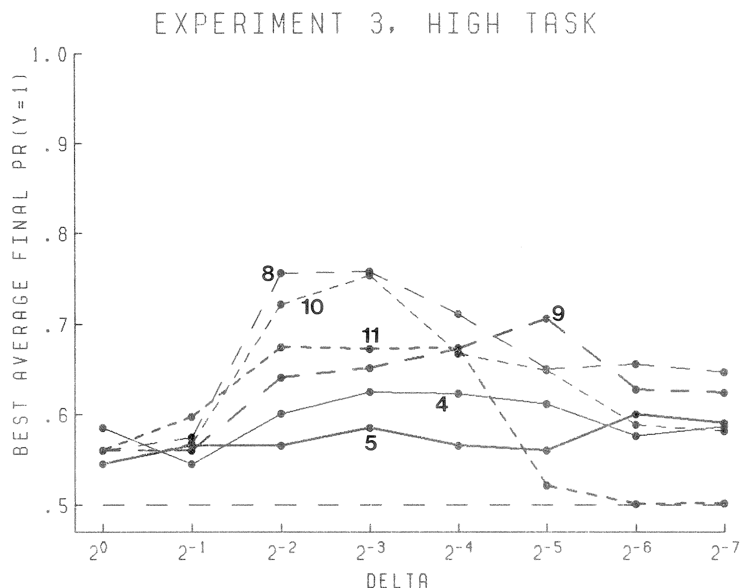


Figure 28. Algorithm Performance on the *High* Task. Each point represents the best performance of a particular algorithm with a particular δ value. Points due to the same algorithm (with different δ values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the chance performance level.

All the reinforcement-comparison algorithms performed better than all the non-reinforcement-comparison algorithms on the high and low tasks. On the middle task, Algorithm 5, a non-reinforcement-comparison algorithm, performed better than reinforcement-comparison Algorithms 8 and 10. On this task, Algorithm 5 and the other two reinforcement-comparison algorithms, Algorithms 9 and 11, all performed at the optimal level, so that any performance difference between them was lost in a ceiling effect.

Recall that with $\delta = 1$, all algorithms become identical to those developed for immediate-reinforcement tasks and studied in preceding chapters. The good per-

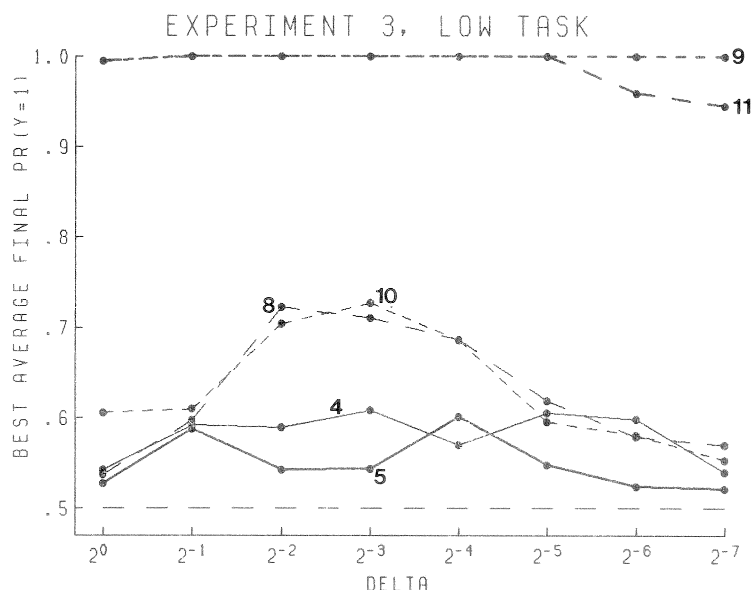


Figure 29. Algorithm Performance on the *Low* Task. Each point represents the best performance of a particular algorithm with a particular δ value. Points due to the same algorithm (with different δ values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the chance performance level.

formance of the reinforcement-comparison algorithms, together with the fact that in all cases their best performance was attained at a δ value less than 1, provide evidence that the new reinforcement-comparison algorithms have been generalized properly for application to delayed-reinforcement tasks.

The performances of the algorithms using $y[t] - \pi[t]$ in their eligibility factors (Algorithms 5, 9, and 11), as contrasted to those using $y[t] - \frac{1}{2}$ (Algorithms 4, 8, and 10), were generally similar to that found in the experiments of Chapter II. On the low and middle tasks those algorithms using $y[t] - \pi[t]$ were, in most cases, clearly superior. On the high task, the $y[t] - \frac{1}{2}$ algorithms were sometimes clearly best. This pattern is similar to that found in Chapter II with immediate-

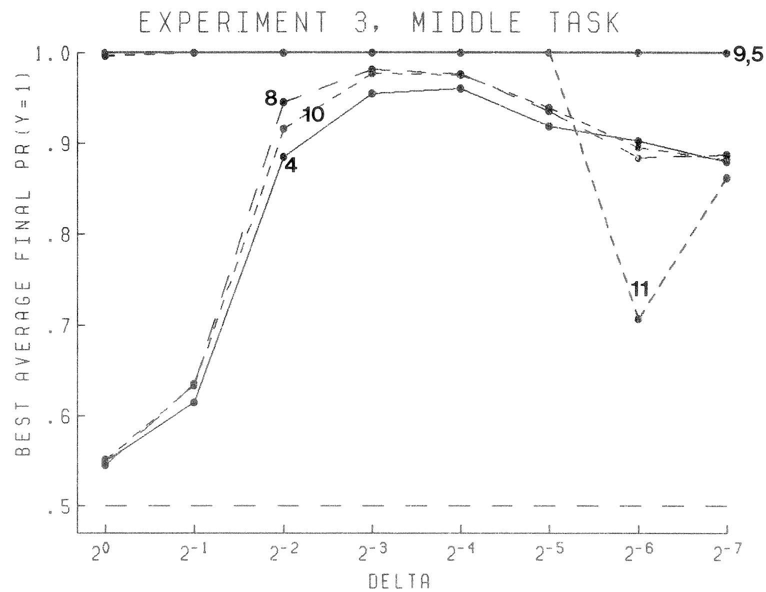


Figure 30. Algorithm Performance on the *Middle* Task. Each point represents the best performance of a particular algorithm with a particular δ value. Points due to the same algorithm (with different δ values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the chance performance level.

reinforcement tasks. In both cases the combination of a reinforcement-comparison mechanism and $y[t]-\pi[t]$ eligibility tended to improve performance in most cases.

In earlier experiments a test of statistical significance was applied to the differences between the best performance levels of each algorithm. Applying such a test to the data of this experiment would be extremely questionable, and it has not been done. In this experiment, two parameters were varied, α and δ, whereas earlier experiments varied only one. To select the best performance level in this experiment would involve a selection of the best from 104 levels, one for each combination of values for α and δ. Such a selection process, when carried out on such

a scale, and when not taken into account in the statistical test, would make the outcome of that test practically meaningless.

Experiment 4: Associative-Delay Asymmetry

The experiments of Chapter III measured the influence on performance of four different kinds of asymmetry in the treatment of stimuli on associative-learning tasks and attempted to relate performance differences among algorithms to features of those algorithms. Delayed reinforcement creates the possibility for another type of asymmetry between stimuli: reinforcement for an action chosen in response to one stimulus may be delayed more than reinforcement for an action chosen in response to a second stimulus. Experiment 4 adds this ***associative-delay asymmetry*** to those of the preceding chapter.

Associative-delay asymmetry is not to be confused with the asymmetry investigated in Experiment 2 of this chapter. The asymmetry in Experiment 2 is an *action* asymmetry rather than a *stimulus* asymmetry. In Experiment 2's asymmetrical task, the actions are treated differently, whereas in Experiment 4's task, as in the tasks of Chapter III, it is the stimuli that are treated asymmetrically.

Like most of the tasks of Chapter III, Experiment 4's task involved two stimuli presented at random with equal frequency. The two stimuli are similar and yet have different optimal actions. The two stimuli are

$$\vec{x}^1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \qquad \text{and} \qquad \vec{x}^2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}.$$

Actions chosen in response to $\vec{x}^2$ influence reinforcement received on the next step (i.e., immediately, no delay), and actions chosen in response to $\vec{x}^1$ influence

reinforcement received after a delay of 4 steps. In either case, actions have either a $+.1$ influence on subsequent reinforcement or a $-.1$ influence. In response to $\vec{x}^2$, Action 0 is the action with the positive influence; in response to $\vec{x}^1$, Action 1 is the action with the positive influence. If influences from two actions in response to two different stimuli both influence the same reinforcement value, their influences sum, e.g., if $y[t] = 0$ is selected in response to $\vec{x}[t] = \vec{x}^2$, and $y[t-4] = 1$ is selected in response to $\vec{x}[t-4] = \vec{x}^1$, then $r[t+1]$ is $+.2$. If a reinforcement is not influenced by any previous action, it is zero. In this task reinforcement is computed deterministically from the actions selected.

Simulations were run with all 6 algorithms with each combination of a range of values for the parameters α and δ. The α values used were the powers of 2 from 2^{-5} to 2^6, and the δ values used were the powers of 2 from 2^0 to 2^{-7}. For each algorithm and parameter setting, 200 runs of exactly 150 steps were simulated. At the end of each run, the final weight vector $\vec{w}[151]$ was recorded. From this, the probability of selecting each action in response to each stimulus and the expected influence on reinforcement of the next action, was computed as described in Chapter III. The expected influence of the next action on subsequent reinforcement is a good measure of performance on a particular run. This measure was averaged over the 200 runs to produce a performance measure for each algorithm with each parameter setting.

Figure 31 summarizes the data from Experiment 4, showing only the performance levels for the α value associated with maximum performance for each algorithm and δ value. Algorithms 9 and 11 performed best, and were particularly superior at low δ values. Apparently, both reinforcement comparison and the use of an eligibility factor including $y[t] - \pi[t]$, as opposed to $y[t] - \frac{1}{2}$, are necessary for maximal performance on this task.

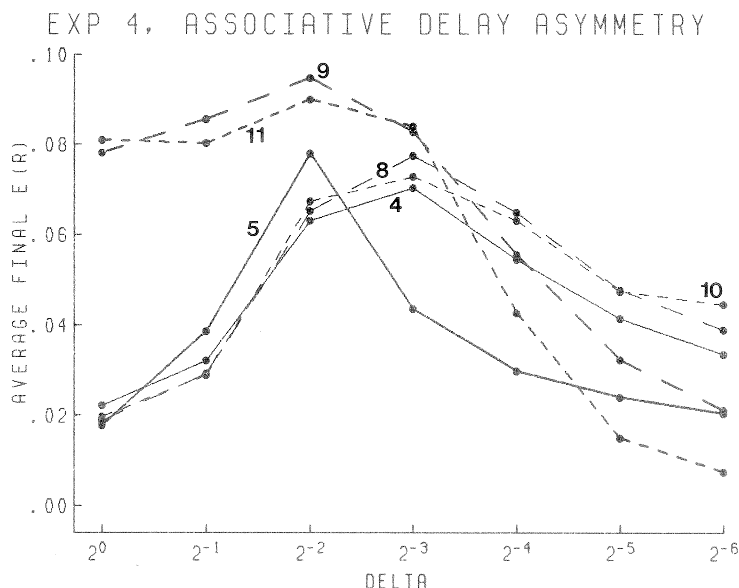


Figure 31. Summary of Algorithm Performance on a Task with Associative-Delay Asymmetry. Each point represents the average performance level of a particular algorithm with a particular eligibility trace length (δ value), and with the α value at which performance was best for that algorithm. Points due to the same algorithm (with different δ values) are connected by lines; the numeric label indicates the associated algorithm. The horizontal dashed line indicates the chance performance level.

Conclusions

Reinforcement-learning tasks with delayed reinforcement are much more difficult than corresponding tasks with immediate reinforcement because of the uncertainty introduced as to which of the past actions caused each reinforcing event. The algorithms studied in previous chapters were designed for immediate-reinforcement tasks and are not effective on delayed-reinforcement tasks. However, by adding exponentially-decaying eligibility traces, these algorithms can be modified so as to

greatly improve their performance on delayed-reinforcement tasks. In selecting the length of the traces one must accept a trade-off between performance on short-delay tasks and performance on long-delay tasks. Trace durations appropriate for long delays result in suboptimal learning rates on tasks with short delays. Whatever trace durations are used, learning is always slower with longer delays than with shorter delays. If the delay length were known or could be learned, this trade-off could be avoided.

For algorithms with eligibility traces based on a recency heuristic, delayed reinforcement is less effective than immediate or less-delayed reinforcement. With such algorithms, a small immediate reinforcement may be more effective in reinforcing behavior than a larger delayed reinforcement. Experiment 2 shows that all algorithms considered here, even those using reinforcement-comparison techniques and $y[t]-\pi[t]$, are susceptible to this problem. On nonassociative tasks with equal delays for both actions, the algorithms introduced in this chapter perform similarly to the algorithms of Chapter II on immediate-reinforcement tasks. Overall, there are advantages to both reinforcement-comparison mechanisms and to the use of a $y[t]-\pi[t]$ eligibility mechanism.

Finally, delayed reinforcement opens the possibility of a new type of asymmetry between the treatment of stimuli in associative learning — the delay between action and reinforcement may be different for actions in response to different stimuli. Experiment 4 contrasted the ability of the various algorithms to deal with this asymmetry on a task analogous to those studied in Chapter III. As in Chapter III, reinforcement-comparison and $y[t]-\pi[t]$ algorithms performed best.

Although delayed reinforcement is a necessary property of any task with an opportunity for secondary reinforcement, the tasks of this chapter were carefully designed to offer no such opportunities. All tasks of this chapter either had no stimuli or else stimuli that were presented at random. Opportunities for secondary

reinforcement to be useful arise only when actions of the learning system affect subsequent stimuli. These studies of delayed reinforcement without secondary reinforcement provide a point of departure for the study of secondary reinforcement.

CHAPTER V

SECONDARY-REINFORCEMENT ALGORITHMS

In this chapter several temporal credit-assignment algorithms are discussed that involve secondary reinforcement, i.e., that associate stimuli with forthcoming reinforcement and then use the information provided by their occurrence to better assign temporal credit. Among the algorithms discussed are a version of the algorithm used by Samuel in his checker-playing program (Samuel, 1959), that used by Witten in his adaptive controller (Witten, 1977), and a new algorithm called the adaptive heuristic critic (AHC) algorithm. The algorithms and the problems they solve are related to each other via the theoretical concept of an ideal reinforcement signal, discussed in the first section below. In the final section of this chapter, a range of apparently different algorithms are proven to all be equivalent to the AHC algorithm by a choice of constants and rearrangement of terms.

Although not discussed here, related work has also been done by Holland (in preparation), Booker (1982), and Hampson and Kibler (1982). Holland's "bucket brigade" appears to be closely related to the AHC algorithm, but is used within a much more complex context. Booker has investigated the use of secondary reinforcement algorithms within the context of Holland's "classifier systems" and "genetic algorithm." Hampson and Kibler have considered secondary-reinforcement mechanisms along the lines of Samuel's work for use in associative reinforcement learning.

The Ideal Reinforcement Signal

Reinforcement signals can vary in the quality of the evaluative information they provide to the learning system. As has been seen in previous chapters, they can be noisy, delayed, and unbalanced in their distribution of positive and negative values. One can assume that in most problems the primary reinforcement signal is of low quality in at least one of these respects. If one considers all the possible ways in which a reinforcement signal can be of low quality, and then imagines a signal that is ideal in all those respects, then one has the concept of the ideal reinforcement signal.

The *ideal reinforcement signal* is positive whenever the immediately preceding action is better than average for the situation in which it was taken, and negative whenever that action is worse than average. Its magnitude indicates how much better or worse than average the immediately preceding action is. By the *value* of an action, I mean a measure of its expected effect on future values of the primary reinforcement signal. The ideal reinforcement signal indicates the value of the selected action relative to the average of the values of all possible actions, each weighted according to its probability of being chosen in the given situation. Since these frequencies depend on the current state of the learning system, so does this average, and consequently the ideal reinforcement signal also depends on the state of the learning system. In fact, since the ultimate effect of an action on primary reinforcement is frequently dependent on subsequent actions, the value of an action in general also depends on the state of the learning system. Since the learning system changes state as it accumulates experience, the ideal reinforcement signal also changes.

For example, suppose there is a task with 3 actions, with the values 0, 9, and 10 when taken in a particular situation by a particular learning system. In the early stages of learning, when the 3 actions are each selected equally often, the

ideal reinforcement signal will be positive for the latter two actions, and negative for the first. As learning progresses, and the probability of selecting the first action falls to zero, the ideal reinforcement signal will change so as to be negative for the second action, remaining positive only for the third. One could argue that in order to satisfy one sense of the word "ideal," the ideal reinforcement signal should be a constant function of the actions and should always be positive only for the best of the actions, in this case only for the third action. Such a reinforcement signal might cause *perfect* performance to be attained more quickly, but probably only at the cost of sacrificing interim performance. The ideal reinforcement signal should stimulate the maximization of *cumulative* performance.

For example, consider a chess-playing task. Suppose a position has been reached from which the learning system can force checkmate by playing move A followed by move B, but that if it plays A and then overlooks B, it will lose the game. Suppose further that with the learning system's current state, the latter is exactly what will happen if A is chosen. Finally, suppose that if the learning system chooses some move other than A, then it has a better than even chance of winning the game.

In one sense A is a good move in this position, since it is the first move in a sequence of moves that guarantees a win. On the other hand, with the current state of the learning system, A is a terrible move, because it results in the immediate loss of a game that might otherwise be won. Here the value of a particular move is defined in the latter, local sense that would label A a poor move. The idea of the ideal reinforcement signal is to consider each move in isolation from all other factors and ask "Other things left as they are, does selecting this move make the prospects better or worse?"

The ideal reinforcement signal improves over the primary reinforcement signal in three ways. These are, in decreasing order of importance to the work presented

here:

Immediacy — Some effects of the choice of an action on subsequent primary reinforcement may be delayed. In the ideal reinforcement signal all delayed effects would be "brought into the present," so that they are effective immediately after the action is taken. Immediate reinforcement is an improvement over delayed reinforcement because there is no uncertainty as to which action causes it.

Comparison with a reinforcement standard — A given reinforcement level is not good or bad in itself, but only in comparison with other reinforcement levels that might have been received had the learning system or the environment behaved differently. Since the ideal reinforcement signal rates actions that are better or worse than average as being positive and negative, it provides direct information as to whether the selected action is a good or a bad selection. The ideal reinforcement signal removes from the learning system the burden of comparing reinforcements with a ***reinforcement standard*** and for determining what that standard should be.

Increased reliability – The ideal reinforcement signal is a relative measure of how good or how bad the selected action is *on the average.* Whereas the primary reinforcement signal might be different each time a particular action is taken in a particular situation, due to random aspects of the environment, or to variations in subsequent action selections, the ideal reinforcement signal provides a reliable measure that takes into account all possible variations and their likelihoods.

The concept of the ideal reinforcement signal is similar to that of the ideal evaluation function for guiding a state-space search or the search through a game tree. In either case the ideal is rarely obtainable but can only be approximated. In either case one looks to heuristics, either learned or provided *a priori*, to build the approximation. A ***heuristic reinforcement signal*** is a reinforcement signal generated internally by a reinforcement-learning system, which it uses instead of the primary reinforcement signal. The idea is that the heuristic reinforcement signal is more

similar to the ideal reinforcement signal than is the primary reinforcement signal. To improve on the primary reinforcement signal, a learning system may use either *a priori* knowledge or knowledge gained from its past experience. In the present work only the latter is considered; it is assumed that all available *a priori* knowledge has been built directly into the primary reinforcement signal.

Given the limited information a learning system receives about the state of its environment, how good a heuristic reinforcement signal is actually possible? The *ideal realizable reinforcement signal* is positive at the first indications from the environment that higher than average primary reinforcement is forthcoming, and negative at the first indications that lower than average primary reinforcement is forthcoming. To the extent that the environment provides such information, the size of the signal indicates the amount by which the forthcoming reinforcement is higher or lower than average. It is the ideal realizable reinforcement signal that one can most fruitfully attempt to emulate by a heuristic reinforcement signal.

In some cases the first indication of forthcoming primary reinforcement is the primary reinforcement itself. In this case the ideal realizable reinforcement signal is an improvement over the primary signal only by virtue of providing a performance standard, and not by improving immediacy or reliability. In other cases, stimuli provide the first indications of unusually high or low forthcoming reinforcement, and improvements in immediacy are realizable as well.

Learning algorithms in which neutral stimuli are recognized as cues to forthcoming primary reinforcement, and are used to create more immediate heuristic reinforcement, are called *secondary reinforcement algorithms.* In the following sections it is shown how secondary reinforcement algorithms can be informally derived as approximations to the ideal reinforcement signal.

The Adaptive Heuristic Critic Algorithm

The adaptive heuristic critic (AHC) algorithm is a heuristic reinforcement or "critic" algorithm incorporating secondary reinforcement. The AHC algorithm is closely related to a model of classical conditioning (Sutton and Barto, 1981; Barto and Sutton, 1982). The development of the AHC algorithm, particularly my early work in this area (Sutton, 1978), has been influenced by the work of Klopf (1972, 1980).

In the AHC algorithm, a linear-mapping approach is used to associate predictions of forthcoming reinforcement with stimuli. Stimuli are represented as vectors of n components. Associated with each stimulus component x_i, $1 \leq i \leq n$, is a memory variable v_i indicating the extent to which the presence of the stimulus component indicates that unusually high or low reinforcement is forthcoming. As in Chapters 3 and 4, the vector $\vec{v}[t] = (v_1[t], v_2[t], \ldots, v_n[t])$ is called the *reinforcement-association vector.* The *prediction of reinforcement forthcoming after time* t made by the learning system at time s (i.e., using $\vec{v}[s]$) is denoted by $p^s[t]$ and defined as:

$$p^s[t] = \sum_{i=1}^{n} v_i[s]x_i[t]. \tag{22}$$

The reinforcement-association vector should be learned such that $p^s[t]$ becomes a good estimate of forthcoming reinforcement. It follows, therefore, that a signal that is a good heuristic reinforcement signal would provide an excellent basis for correcting estimates of $p^s[t]$ by changing $\vec{v}$. The following update rule accomplishes this:

$$v_i[t+1] = v_i[t] + \beta\, \hat{r}[t+1]\bar{x}_i[t], \qquad v_i[0] = 0, \tag{23}$$

for $1 \leq i \leq n$ and $t = 0, 1, \ldots,$ where $\hat{r}[t+1]$ is the heuristic reinforcement signal's value at time $t+1$, β is a positive constant, and $\bar{x}_i[t]$ denotes the value

at t of a trace of x_i, i.e., a weighted average of the values of x_i with the more recent values weighted more heavily (see Chapter IV, Equation 15). If $\hat{r}[t+1]$ is positive (negative) this equation increases (decreases) the components of $\vec{v}$ that contributed to past predictions $p^e[t]$, as indicated by their $\vec{x}$ components having been large in the recent past. If the same stimulus sequence is presented again, this process results in predictions being higher (lower) earlier in the sequence. The overall result is that the positive (negative) $\hat{r}$ event is shifted earlier in time, and that the heuristic reinforcement signal does a better job of giving the earliest possible indication of changes in forthcoming reinforcement.

The above argument relies on the heuristic reinforcement signal working properly. The following definition of the heuristic reinforcement signal is that used in the AHC algorithm:

$$\hat{r}[t+1] = r[t+1] + \gamma\, p^t[t+1] - p^t[t], \tag{24}$$

where γ, $0 \leq \gamma \leq 1$, is a scalar *discount rate parameter.* Equations 22, 23, and 24 (with the trace operator denoted by "$^-$" and defined by (15)) constitute the AHC algorithm in its entirety.

The remainder of this section explores the relationship between the heuristic reinforcement signal defined by (24) and the concept of the ideal reinforcement signal. Let r^* denote the ideal reinforcement signal. Its value at time $t+1$ indicates how much better or worse off the learning system is because of the particular action selected at time t. Better or worse is defined in terms of the effect of the action on subsequent primary reinforcement signal values $r[t+k]$, $k \geq 1$. The action $y[t]$ at time t may affect primary reinforcement at any combination of later times. These separate effects must somehow be combined in r^* to give an overall measure of the action's effect. Perhaps the simplest approach would be to sum these effects if they could be determined. This leads one to attempt to define the

ideal reinforcement signal as

$$r^*[t+1] = \sum_{k=1}^{\infty} \Big[E\{r[t+k] \mid y[t]\} - E\{r[t+k]\} \Big], \tag{25}$$

where both expectations are also conditional on the structure and state of the environment and of the learning system. One problem with (25) is that the infinite sum may not be convergent. Another is that although this definition seems a natural one, for certain special classes of tasks it is not. In the following section the class of ***time-blind tasks*** is considered, for which a different definition of r^* is appropriate.

The sum in (25) may not converge either because it is unbounded (infinite) or because the sequence of partial sums includes infinite subsequences not all of which converge to the same limit. Figure 32a shows the state-transition structure of an environment in which the former occurs, and Figure 32b shows an environment in which the latter occurs, for the action selected in leaving the state labeled A. These cases are problematic for the definition in (25) but seem not to be problematic on an intuitive basis. The reinforcement given to the action leaving State A is unimportant because the state is never reentered. In many cases these definitional problems can be eliminated by weakening the requirement of convergence of (25) to that of summability. In general it may be necessary to make a slightly different definition of r^* depending on the class of tasks or learning systems being considered. For the purposes of the rest of this section, it is assumed that the sum in (25) is convergent.

The sequence formed by $E\{r[t+k] \mid y[t]\} - E\{r[t+k]\}$ for successive values of k is called the ***difference sequence***. Each element of this sequence is the difference between the expected value of reinforcement at some later time before and after the selection of $y[t]$. In a typical case $y[t]$ might influence subsequent reinforcement and the state of the environment for a while, but for large values of k the two expectation would be equal, and the elements of the difference sequence zero. In

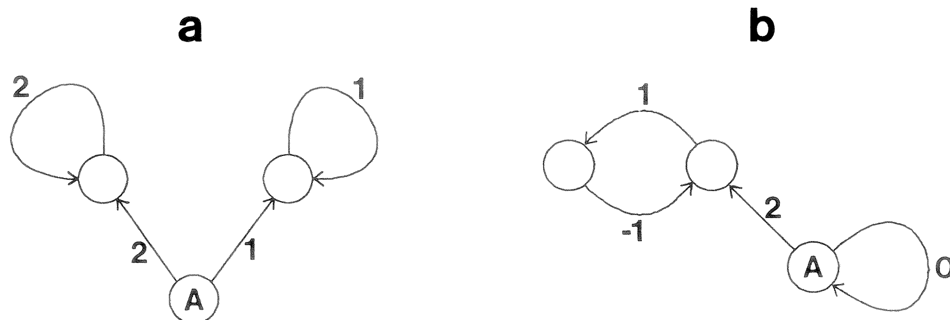


Figure 32. State-Transition Structure of two Environments that are Problematic for Definitions of the Ideal Reinforcement Signal. Each circle represents a state of the environment. The state labeled "A" is the initial state. Arc labels indicate the expected value of the primary reinforcement signal should that state transition occur. Which state transition occurs (which arc is followed) depends deterministically on the action selected by the learning system. Where only one arc leaves a state all actions result in the same state transition.

such a case the sum of all elements of the sequence is finite, and its value is the value of r^*.

One possible approach to approximating r^* is to maintain a memory variable whose value is the current estimate of the sum of the difference sequence. A major drawback to this approach is that these differences are never directly observable. Each is the difference between two expected values of $r[t+k]$, both for some time $t+k$, but conditional on different circumstances. The actual values of r provide an estimate of the expected value for the circumstances that actually occur, but the other circumstances remain hypothetical. A second drawback to this approach is that an estimate of the sum of a difference sequence would be needed for every combination of action and environmental state in which the action might be taken, which adds up to a great many memory variables to update and store.

Another approach, used in the AHC algorithm, is to estimate the sum of the difference sequence by constructing separate estimates of the sum of the

$E\{r[t+k] \mid y[t]\}$ terms and the sum of the $E\{r[t+k]\}$ terms, and then to subtract the two estimates. The logic of this approach can be illustrated by rewriting (25) as a difference of two sums:

$$r^*[t+1] = \sum_{k=1}^{\infty} E\{r[t+k] \mid y[t]\} - \sum_{k=1}^{\infty} E\{r[t+k]\}. \tag{26}$$

In this approach, one attempts to estimate both terms of (26) from observed values of r, and then subtract the two estimates to estimate r^*. Unfortunately, (25) and (26) are not equivalent. Although (25) is typically (and by assumption) finite, the two sums of (26) are typically both infinite (divergent). Making finite estimates of these two and then subtracting them yields a meaningless result.

The AHC algorithm's solution to this problem relies on the fact that the elements of the difference sequence tend to be nonzero primarily only if they appear early in the sequence, or, stated in different terms, that for large k the expected value of $r[t+k]$ tends to depend very little on $y[t]$. If it is assumed that

$$E\{r[t+k] \mid y[t]\} = E\{r[t+k]\},$$

for all k greater than M, then (25) can be rewritten as

$$\begin{aligned} r^*[t+1] &= \sum_{k=1}^{M} \Big[E\{r[t+k \mid y[t]\} - E\{r[t+k\} \Big] \\ &= \sum_{k=1}^{M} E\{r[t+k] \mid y[t]\} - \sum_{k=1}^{M} E\{r[t+k]\}. \end{aligned} \tag{27}$$

The advantage of (27) is that, unlike (26), both of its sums are finite and can thus be estimated.

The AHC algorithm actually works a bit differently, but the idea is the same. Rather than having an abrupt cutoff at M time steps, successive elements of the difference sequence are weighted in an exponentially decreasing manner. This

avoids problems with "horizon effects," and allows arbitrarily late elements of the difference sequence to contribute to the estimate of r^*, albeit possibly greatly discounted. The AHC algorithm is based on estimating

$$\begin{aligned} r^*[t+1] &\approx \sum_{k=1}^{\infty} \gamma^{k-1} \Big[E\{r[t+k] \mid y[t]\} - E\{r[t+k]\} \Big] \\ &= \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid y[t]\} - \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k]\}, \end{aligned} \tag{28}$$

where $0 \leq \gamma < 1$. These sums converge as long as $E\{r[t+k]\}$ is bounded.

γ is called the ***discount rate.*** The use of a discount rate effectively assigns greater value to earlier primary reinforcement than later primary reinforcement. It is as if instead of maximizing the sum of future reinforcements, one maximizes the sum of future reinforcements weighted in an exponentially decreasing fashion according to the time delay until their reception:

$$\sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k]\} \tag{29}$$

By adjusting γ, one controls the extent to which the learning system is concerned with long-term versus short-term goals. It is common in studying Markovian decision processes to introduce such a discount rate by taking the maximization of (29) directly as the goal of learning rather than as an approximation (e.g., Derman, 1970; Mine and Osaki, 1970).

The expected values discussed thus far are all implicitly conditional on the structure and state of the environment and the learning system. In the following it is necessary to forego this notational convenience in the case of the states. Denoting the state of the environment at time t as $q[t]$ and the complete state of the learning

system at time t as $w[t]$ *, (28) can be rewritten as:

$$r^*[t+1] \approx \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid y[t], q[t], w[t]\} - \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid q[t], w[t]\}. \tag{30}$$

At time $t+1$, $r[t+1]$ will be available for use as an estimate of $E\{r[t+1] \mid y[t], q[t], w[t]\}$. Using this approximation, and taking this term outside the first sum in (30) yields

$$\begin{aligned} r^*[t+1] \approx r[t+1] &+ \sum_{k=2}^{\infty} \gamma^{k-1} E\{r[t+k] \mid y[t], q[t], w[t]\} \\ &- \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid q[t], w[t]\}. \end{aligned}$$

Since $r[t+k]$, for $k \geq 2$, does not depend on $y[t]$ directly, but only through $y[t]$'s effect on $q[t+1]$, the first expected value in the above expression can be rewritten to be conditional only on $q[t+1]$:

$$\begin{aligned} r^*[t+1] \approx r[t+1] &+ \sum_{k=2}^{\infty} \gamma^{k-1} E\{r[t+k] \mid q[t+1], w[t]\} \\ &- \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid q[t], w[t]\}, \end{aligned} \tag{31}$$

or, making the change of variable $k \rightarrow k+1$ in the first sum:

$$\begin{aligned} r^*[t+1] \approx r[t+1] &+ \gamma \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+1+k] \mid q[t+1], w[t]\} \\ &- \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid q[t], w[t]\}. \end{aligned}$$

---

* Note that in this context $w[t]$ denotes the *complete* state of the learning system. In the case of the AHC algorithm, this includes both the action-association vector $\vec{w}$ and the reinforcement-association vector $\vec{v}$.

As mentioned above, the AHC algorithm directly constructs estimates of the two sums in these equations. However, both sums involve expectations that are conditional on the state of the environment, which is known only imperfectly through the cues provided by stimuli. The best a *realizable* heuristic-reinforcement algorithm can do is make approximations based on these cues. Denoting by $x[t]$ the stimulus received at time t, which provides information about the state of the environment $q[t]$, the best a realizable heuristic reinforcement signal can do is approximate:

$$\begin{aligned} r^*[t+1] \approx r[t+1] &+ \gamma \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+1+k] \mid x[t+1], w[t]\} \\ &- \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid x[t], w[t]\}. \end{aligned} \tag{32}$$

$p^s[t]$, defined by (22), is the AHC algorithm's estimate at time s of forthcoming discounted reinforcement:

$$p^s[t] \approx \sum_{k=1}^{\infty} \gamma^{k-1} E\{r[t+k] \mid x[t], w[t]\}. \tag{33}$$

In order to avoid instabilities due to secondary reinforcement being able to create more secondary reinforcement, special care must be taken to separate the time s at which the estimate is made from the time t *about* which the estimate is made. Since w may change at each time step, the expected value in (33) may change at each time step. Since the learning system's estimate $p^s[t]$ is based on the statistics of past observations, it will always be slightly out-of-date. For a statistical approach to work, the effect of changes in w on the expected values in (33) must be small, at least over short time periods. If changes in w over small time periods, such as those between t and $t+1$, are ignored, then, using the estimate (33), (32) can be approximated by

$$r^*[t+1] \approx r[t+1] + \gamma\, p^t[t+1] - p^t[t],$$

which is the expression used in the AHC algorithm for its heuristic reinforcement signal $\hat{r}[t+1]$ (cf. (24)).

The AHC algorithm uses the linear-mapping approach given by (22) to construct its estimates $p^e[t]$, and uses $\hat{r}$ as an error term to update its estimates by (23). The logic of viewing $\hat{r}$ as giving the error in estimating forthcoming reinforcement (i.e., in $p^e[t]$) is best seen from (31) and (33). If (33) is exact for both $p^t[t+1]$ and $p^t[t]$, then (31) is zero. If it is non-zero, it indicates that the earlier prediction may be too large or small and should be adjusted accordingly.

Time-Blind Tasks

Further understanding of the AHC algorithm and of the informal derivation presented above can be gained by performing a similar analysis for other classes of tasks. This and the following two sections perform such an analysis for slightly different classes of tasks. In many cases the steps of a derivation are nearly identical to that of preceding derivations, in which case most of the justification is omitted.

This section presents an informal derivation of a heuristic reinforcement algorithm that approximates an ideal reinforcement signal for the class of "time-blind tasks," which includes most board games. For this class of tasks, the analysis yields an algorithm closely related to the "learning-by-generalization" algorithm used by Samuel in his checker-playing program (Samuel, 1959, 1967).

In many tasks, most prominently in games such as chess or checkers, the environmental interaction can be naturally divided into episodes, where the performance during each episode is dependent only on the behavior during the episode, and where the boundaries between episodes are clearly demarcated. In this dissertation such tasks are called *episodic tasks.* In a game such as chess or backgammon,

for example, the episodes are games. In the pole-balancing task considered later in this chapter, each attempt to balance the pole, from its initial position to its falling over, is an episode. Any task in which the environment repeatedly returns to a start state, and in which the learning system is given unambiguous notice of its return, can be considered an episodic task.

Whether or not a task is episodic can be very important for purposes of temporal credit assignment. At the end of each episode the learning system is guaranteed that there will be no further delayed effects of any of the actions taken during the episode. Normally this is not possible; the learning system can never "close the books" on the credit to be assigned to any of its past behavior, for there may always be some further consequences. It is the possibility of such arbitrarily long delayed effects in other cases that complicates the definition and approximation of the ideal reinforcement signal.

Time-blind tasks are a particular kind of episodic task in which the goal of learning is defined in terms of performance per episode rather than in terms of performance per time step. In a time-blind task, the goal of learning is completely "blind" to the duration of episodes. Most games are time-blind tasks; in chess or checkers, for example, there is, at least in theory, no concern about the length of each game, but only about its eventual outcome. No additional points are awarded for winning quickly or losing slowly.

The simple environment whose state-transition structure is shown in Figure 33 illustrates the difference between time-blind and non-time-blind tasks. The state marked A is the initial state of each episode. According to a non-time-blind goal of learning, the right path from state A is preferable, since it results in a reinforcement *per time step* of 3, as opposed to 2 for the left path. According to a time-blind goal, on the other hand, the left path from state A is preferable, since it results in a total reinforcement *per episode* of $2+2=4$, as opposed to 3 for the right path.

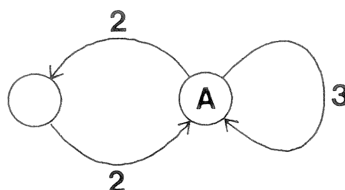


Figure 33. An Environment that is very Different as a Time-Blind and as a Non-Time-Blind Task. Each circle represents a state of the environment. The state labeled "A" is the initial state. Arc labels indicate the expected value of the primary reinforcement signal should that state transition occur. Which state transition occurs (which arc is followed) depends deterministically on the action selected by the learning system. Where only one arc leaves a state all actions result in the same transition.

The goal of learning is different in time-blind and non-time-blind tasks.

In many time-blind tasks each possible outcome of an episode can be assigned a definite value, e.g., $+1$, -1, and 0, for the win, loss, and draw of a chess game. This framework can be made slightly more general by allowing reinforcement to be delivered throughout the episode, where the goal of learning is to maximize the expected value of the sum of the reinforcement received during the episode (this is the framework just used above to analyze the environment shown in Figure 33):

$$E\left\{\sum_{t=1}^{m} r[t]\right\},$$

where $r[t]$ denotes the reinforcement received on the tth time step of the episode, and m denotes the length in time steps of the episode. This expectation is implicitly conditional on the task, the learning system, and the state of the learning system. Tasks in which the outcome of the episode is known only at its end are easily handled within this framework by letting the reinforcement be zero for all transitions except the final one.

Although it applies to time-blind tasks in general, the following analysis is framed in the language of a board game. That is, episodes are called games, the actions of the learning system are called moves, and the states of the environment are called board positions.

In a game that has already gone on for $t-1$ moves and has reached position $q[t]$, the desirability or value of the selected move $y[t]$ is defined as the expected value of the sum of the reinforcement received during the rest of the game:

$$E\Big\{ \sum_{\tau=t+1}^{m} r[\tau] \;\Big|\; y[t], q[t], w[t] \Big\}, \tag{34}$$

where $w[t]$ denotes the state of the learning system and m denotes the number of moves in the game.

The average of the values of the actions (moves) that could have been taken, each weighted by its likelihood of being selected, is:

$$\sum_{y} P\,\{y[t] = y \mid q[t], w[t]\}\; E\Big\{ \sum_{\tau=t+1}^{m} r[\tau] \;\Big|\; y, q[t], w[t] \Big\}.$$

This is mathematically equivalent to the expected value of the outcome of the game, given $q[t]$ and $w[t]$, but without specifying any action:

$$E\Big\{ \sum_{\tau=t+1}^{m} r[\tau] \;\Big|\; q[t], w[t] \Big\}. \tag{35}$$

This expression is called the value of the position $q[t]$.

Using (34) and (35) an exact expression for the ideal reinforcement signal for this class of tasks can be written:

$$r^*[t+1] = E\Big\{ \sum_{\tau=t+1}^{m} r[\tau] \;\Big|\; y[t], q[t], w[t] \Big\} - E\Big\{ \sum_{\tau=t+1}^{m} r[\tau] \;\Big|\; q[t], w[t] \Big\}.$$

The following develops an approximation to r^* by following steps analogous to those used in the preceding section in approximating (28). Since the logic of each

step is the same in both cases, their justification is not repeated here in its entirety. The value of the move $y[t]$ (the first expected value in the above equation) can be approximated by the value of the position $q[t+1]$ that results after making it, plus the reinforcement received traversing from $q[t]$ to $q[t+1]$:

$$r^*[t+1] \approx r[t+1] + E\Big\{ \sum_{\tau=t+2}^{m} r[\tau] \,\Big|\, q[t+1], w[t] \Big\} - E\Big\{ \sum_{\tau=t+1}^{m} r[\tau] \,\Big|\, q[t], w[t] \Big\}.$$

Assuming $w[t]$ does not change rapidly (i.e., from move to move), its time index can be ignored and the latter two terms can be rewritten as the change from one move to the next of a single expression:

$$r^*[t+1] \approx r[t+1] + E\Big\{ \sum_{\tau=t+2}^{m} r[\tau] \,\Big|\, q[t+1], w \Big\} - E\Big\{ \sum_{\tau=t+1}^{m} r[\tau] \,\Big|\, q[t], w \Big\}.$$

Since the stimulus x is known to the learning system, but not the state q the best that can be done is to approximate

$$r^*[t+1] \approx r[t+1] + E\Big\{ \sum_{\tau=t+2}^{m} r[\tau] \,\Big|\, x[t+1], w \Big\} - E\Big\{ \sum_{\tau=t+1}^{m} r[\tau] \,\Big|\, x[t], w \Big\}. \tag{36}$$

$p^s[t]$ is the learning system's estimate at time s of the value of the board position at time t:

$$p^s[t] \approx E\Big\{ \sum_{\tau=t+1}^{m} r[\tau] \,\Big|\, x[t], w[t] \Big\}. \tag{37}$$

For the case of $t = m$ (the end of a trial) this expectation need not be estimated because it must be zero. For all other cases estimates must be formed and associated with the stimuli in some way, e.g., by (22) and (23). Finally, using (37), (36) can be approximated by

$$r^*[t+1] \approx \hat{r}[t+1] = r[t+1] + p^t[t+1] - p^t[t]. \tag{38}$$

The algorithm given by (22), (23), and (38) is clearly closely related to the AHC algorithm. The primary difference between the two is the presence of the discount

rate parameter γ in the AHC algorithm. The algorithm given by (22), (23), and (38) is a special case of the AHC algorithm and is designated the ***Simplified Samuel's algorithm***, or SS algorithm, because of its similarity to that used by Samuel in his checker-playing program (Samuel, 1959, 1967). This similarity is discussed further in a later section of this chapter.

Time-until-Failure Tasks

Experiment 4 of Chapter VI concerns the task of controlling a system so as to keep a pole balanced as long as possible subject to certain other constraints. When the pole falls, or one of the other constraints is not satisfied, a failure is said to occur, the system is reset to an initial position, and a new attempt to balance the pole begins. Tasks such as this, in which all episodes end in failure and in which the object is to delay failure as long as possible, are called ***time-until-failure tasks.*** The ideal reinforcement signal for such tasks is

$$r^*[t+1] = E\{m \mid y[t], q[t], w[t]\} - E\{m \mid q[t], w[t]\},$$

where m is the ultimate duration of the current episode, $y[t]$ is the action selected, $q[t]$ the state of the environment, and $w[t]$ the state of the learning system, on the tth time step of the episode. The ideal reinforcement signal can be rewritten in terms of the expected value of the time remaining in the current episode as:

$$r^*[t+1] = E\{m-t \mid y[t], q[t], w[t]\} - E\{m-t \mid q[t], w[t]\}.$$

Following the same steps as for the preceding informal derivations, and omitting details discussed more fully earlier, r^* for time-until-failure tasks can be estimated as follows:

$$r^*[t+1] \approx E\{m-t \mid q[t+1], w[t]\} - E\{m-t \mid q[t], w[t]\}.$$

Dropping the time index on w and considering expectations conditional on stimuli rather than states yields:

$$r^*[t+1] \approx E\{m-t \mid x[t+1], w\} - E\{m-t \mid x[t], w\}.$$

Rewriting in terms of a single expression yields:

$$r^*[t+1] \approx 1 + E\{m-(t+1) \mid x[t+1], w\} - E\{m-t \mid x[t], w\}.$$

To form a heuristic reinforcement signal, $E\{m-t \mid x[t], w\}$ at time t is estimated by $p^t[t]$, yielding

$$r^*[t+1] \approx \hat{r}[t+1] = 1 + p^t[t+1] - p^t[t].$$

For $t = m$, $p^s[t]$ (for $s = t, t+1$) must be 0 and need not be estimated. For other cases, if a linear mapping approach to association is desired, $p^s[t]$ can be computed and updated by (22) and (23).

Time-until-Success Tasks

A *time-until-success* task is one such as that of exiting a maze, in which the goal is to complete each episode (i.e., trip through the maze) as quickly as possible. Time-until-success and time-until-failure tasks are closely related, as are their heuristic reinforcement algorithms. The ideal reinforcement signal for a time-until-success task is

$$r^*[t+1] = E\{-m \mid y[t], q[t], w[t]\} - E\{-m \mid q[t], w[t]\},$$

where m is the duration of the current episode, $y[t]$ is the action selected, $q[t]$ the state of the environment, and $w[t]$ the state of the learning system, on the tth time step of the episode.

r^* can be approximated following the by-now-familiar pattern:

$$\begin{aligned} r^*[t+1] &\approx E\{-(m-t) \mid y[t], x[t], w\} - E\{-(m-t) \mid x[t], w\} \\ &\approx -1 + E\{-m+t+1 \mid x[t+1], w\} - E\{-m+t \mid x[t], w\} \\ &\approx \hat{r}[t+1] = -1 + p^t[t+1] - p^t[t], \end{aligned}$$

where $p^s[t]$ (for $s = t, t+1$) is an estimate at time s of the negative of the time remaining till success:

$$p^s[t] \approx E\{-m+t \mid x[t], w\}.$$

$p^s[m]$ is known to be 0. Otherwise, if a linear mapping approach is desired, $p^s[t]$ can be computed and updated by (22) and (23).

Samuel's "Learning-by-Generalization" Algorithm

This section explores further the relationship between the temporal credit-assignment component of Samuel's checker-playing program (Samuel, 1959, 1967) and the SS algorithm given by (22), (23), and (38).

Samuel's checker-player uses what would now be considered a standard alpha-beta minimax search (e.g., see Barr and Feigenbaum, 1981) to select moves. As a function of experience, the "static" evaluation function used to evaluate terminal board configurations in the search tree is modified. For each board position, a *backed-up* score is computed by considering possible sequences of moves and propagating backward the evaluations of terminal board positions. The difference between the backed-up score for the current position and the static evaluation of the board position on the preceding move is called delta.* Delta's role in Samuel's tree search is analogous to that of heuristic reinforcement $\hat{r}$ in a reinforcement-

* In some versions of Samuel's program, a rudimentary tree search was also used for the evaluation of the preceding board position used in computing delta.

learning process. When delta is positive, the board position has become better than originally appeared to be the case; when delta is negative, the board position has become worse. In either case, the static evaluation function must be changed so that it does a better job anticipating forthcoming evaluations. Samuel called this process "learning-by-generalization."

In Samuel's checker-player, delta is partly due to the change in the board position from the previous position to the current one and partly due to an anticipation of future board positions. The latter is possible in checkers because there is a strong model of the environment—one knows the current position, the effects on it of one's possible actions, and the likely responses of one's opponent. Assuming that no such strong model is available, as has been done in this dissertation, a contribution to delta due to anticipated board positions is not possible. Delta then becomes simply the difference between the static evaluations of the current and preceding board positions.

Samuel's static evaluation function is a sum of terms, each a product of a numerical measure of a feature of the board position and a coefficient indicating the desirability of that feature in terms of its correlation with changes in subsequent evaluations. As a function of experience, the coefficients of the terms are modified, and occasionally new features are added and old ones removed. Here I consider only Samuel's algorithm for modifying the coefficient values. One term, the *piece-advantage term*, does not have a modifiable coefficient. This term measures the number of checkers the program has relative to the number its opponent has, giving a higher weight to kings, and including refinements so that it is considered better to trade pieces when one is ahead but not when one is behind.

Let $F[t]$ denote the value of the piece-advantage term, including its non-modifiable coefficient. Let $x_i[t]$ denote the numerical measure for the ith feature of the board position, and let $v_i[t]$ denote the corresponding modifiable coefficient.

Samuel's static evaluation of the board position occurring at time t can then be written as

$$F[t] + v_1[t]x_1[t] + v_2[t]x_2[t] + \cdots + v_n[t]x_n[t].$$

Let $\delta[t]$ denote the value of Samuel's delta at move t. For the case in which delta has no anticipatory portion,

$$\delta[t+1] = F[t+1] + \sum_{i=1}^{n} v_i[t]x_i[t+1] - F[t] - \sum_{i=1}^{n} v_i[t]x_i[t].$$

Note that both sums use the coefficient values at the same time t. Samuel found this necessary to prevent instability. The above equation can be rewritten, using the $p^t[t]$ notation defined by (22) to denote the sums, as

$$\delta[t+1] = F[t+1] - F[t] + p^t[t+1] - p^t[t].$$

In Samuel's algorithm, changes in the piece-advantage term play the same role as the primary reinforcement signal, i.e., $F[t+1] - F[t]$ is analogous to $r[t]$. $\delta[t+1]$ itself is analogous to $\hat{r}[t+1]$.

In Samuel's words, the idea behind his algorithms is that:

> ...we are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board position of the chain of moves which most probably will occur during actual play. [Samuel, 1959, p. 219]

If this quote is taken literally, it says that the evaluation at time t should be equal to the expected value of the evaluation at any later time $t+k$:

$$F[t] + p^t[t] = E\left\{F[t+k] + p^{t+k}[t+k]\right\}$$

or

$$p^t[t] - E\left\{p^{t+k}[t+k]\right\} = E\{F[t+k]\} - F[t]$$

$$= E\Big\{ \sum_{\tau=t+1}^{t+k} r[\tau] \Big\}$$

(using $r[t] = F[t+1] - F[t]$)

$$= E\Big\{ \sum_{\tau=t+1}^{m} r[\tau] \Big\} - E\Big\{ \sum_{\tau=t+k+1}^{m} r[\tau] \Big\}, \tag{39}$$

where m denotes the number of moves in the game. From (39) it is clear that if (37) holds, then the algorithm accomplishes the result Samuel intends. In other words, Samuel's intent and the intent of the derivation based on the concept of the ideal reinforcement signal for time-blind tasks are the same.

How are the coefficients v_i to be updated so as to implement this intent? In Samuel's words:

> If delta is positive it is reasonable to assume that the initial board evaluation was in error and terms which contributed positively should have been given more weight, while those that contributed negatively should have been given less weight. A converse statement can be made for the case where delta is negative. Presumably, in this case, either the initial board evaluation was incorrect, or a wrong choice of moves was made, and greater weight should be given to terms making negative contributions, with less weight to positive terms. [Samuel, 1959, p. 219]

This is exactly what (23) does, where $\hat{r}$ plays the role of delta. However, the sentence that immediately follows the above quotation is:

> These changes are brought about in an involved manner which will now be described.

Samuel does not implement the algorithm he describes in a direct manner as in (23), but goes through a much more complicated procedure apparently with the aim of arriving at the same end. This procedure involves keeping records that estimate the correlation between the sign of delta and the sign of each $v_i x_i$, normalizing the correlation values so that the largest is a fixed size, taking their ratios, and setting the coefficients to the integral power of 2 whose exponent is closest to the corresponding ratio. In addition, if delta or an evaluation is below an arbitrary

minimum value, it is set to zero, and delta is doubled or quadrupled if its value is due to a change in the piece-advantage term or if that change is particularly large, respectively. In most versions of the checker-player, the coefficient update algorithm involved still further complexities.

Samuel is not explicit about his rationale for making the desired changes in the coefficients in such an involved manner. Perhaps a major reason was the need for a very computation-time-efficient algorithm. If coefficients are powers of two, the evaluation function can be computed much more rapidly on a digital computer than if they are allowed to take on arbitrary values. It is not plain how the algorithm given by (23) could have been implemented so as to keep the coefficients at integral powers of 2.

Witten's Adaptive Controller

Witten (1977) describes an adaptive controller for discrete time Markov environments which uses a collection of learning automata and a temporal credit-assignment algorithm nearly identical to the AHC algorithm. Witten assumes that the environment has discrete states, and that the learning system always knows the current state of the environment. This allows the use of an independent-associations approach to association: For each state i $(1 \leq i \leq n)$ of the environment there is a dedicated memory variable, denoted here as v_i, whose value is the learning system's estimate of future discounted reinforcement (as in (29)).

According to Witten's algorithm, and in terms of his own equations, if the environment moves from state j to state k, then v_j is updated by

$$v_j \leftarrow (1-\beta)v_j + \beta g' \qquad \beta \in (0,1),$$

where g' is a weighted average of the environment's reward g and the controller's

new estimate of future reward:

$$g' = (1-\gamma)g + \gamma v_k,$$

where $\gamma \in (0,1)$ is the discount factor. Witten's g is analogous to primary reinforcement r. Combining the above two equations, using r for g, and adding explicit references to time, where t and $t+1$ are the times at which the environment is in states j and k, yields

$$v_j[t+1] = (1-\beta)v_j[t] + \beta\left((1-\gamma)r[t+1] + \gamma v_k[t]\right).$$

If $\vec{x}[t]$ is used to denote the n-vector all of whose components are zero except for the ith one, which is 1, where the environment is in state i at time t, and $p^e[t]$ is defined by (22), then $p^t[t] = v_j[t]$, $p^t[t+1] = v_k[t]$, and Witten's update equation can be rewritten as

$$v_j[t+1] = v_j[t] + \beta\left((1-\gamma)r[t+1] + \gamma p^t[t+1] - p^t[t]\right).$$

This change in notation converts Witten's algorithm from the independent-associations approach to a degenerate case of the linear-mapping approach. The above equation is only applied to the v_j corresponding to the state j of the environment at time t. Alternatively, $x[t]$ can be used to make this selection, by applying the following equation to all v_i, $1 \leq i \leq 0$:

$$v_i[t+1] = v_i[t] + \beta\left((1-\gamma)r[t+1] + \gamma p^t[t] - p^t[t+1]\right) x_i[t] \qquad \forall i.$$

There are two apparent differences between this equation and the combination of (23) and (24), as in the AHC algorithm. One is that (23) uses a trace $\bar{x}_i[t]$ where the above equation uses $x[t]$ (Note that for the case of $\lambda = 0$, $\bar{x}[t] = x[t]$, so this is not a major difference). The second apparent difference is the presence of the $(1-\gamma)$ multiplier of $r[t+1]$ in the above equation, and its absence in (24).

It is shown in the following section that this, and in fact a much wider range of apparent differences between algorithms, can be eliminated by changing parameter values. The above equation is equivalent to a combination of (23) and (24).

One is tempted to conclude that Witten's algorithm is identical to the AHC algorithm. However, although Witten's algorithm updates the reinforcement-association vector $\vec{v}$ in essentially the same way as does the AHC algorithm, it defines $\hat{r}$ in terms of $\vec{v}$ differently. In Witten's notation, his algorithm uses g' for $\hat{r}$. In the notation used elsewhere in this dissertation, it uses

$$\hat{r}[t+1] = (1-\gamma)r[t+1] + \gamma p^t[t+1]. \tag{40}$$

The complete adaptive controller that Witten describes consists of the heuristic-reinforcement algorithm described above and a bank of learning automata, with this $\hat{r}$ as the interface between them.

In Witten's algorithm, $\hat{r}$ (g') represents an estimate of the sum of forthcoming (discounted) reinforcement. This is in contrast with the AHC and Samuel's algorithm, in which $\hat{r}$ is supposed to estimate the *difference* between forthcoming reinforcement before and after the learning system has committed to a particular action selection. Witten's algorithm includes a secondary-reinforcement mechanism, but not a reinforcement-comparison mechanism.

Equivalent Expressions for the AHC Algorithm

There are many algorithms that involve some of the same ideas of the AHC algorithm, including that of comparing new and old estimates of forthcoming reinforcement, yet they appear to treat these ideas in slightly different ways. Here are two examples:

$$\hat{r}[t+1] = r[t+1] - p^t[t] + \rho(p^t[t+1] - p^t[t])$$

and

$$\hat{r}[t+1] = r[t+1] + \rho(p^t[t+1] - p^t[t]).$$

where ρ is a positive constant and $p^s[t]$ is defined by (22) and (23) in both cases. In other words, both algorithms are similar to the AHC algorithms except that they substitute for its key equation (24) one of the above two equations. In actuality, these two algorithms are both special cases of the AHC algorithm. The following theorem shows that these and a wide class of other algorithms are equivalent to the AHC for particular choices of the parameters of the AHC algorithm, i.e., of α, β, and γ.

Theorem. *For any sequences of real numbers $x_i[t]$, $\bar{x}_i[t]$, $1 \leq i \leq n$, and $r[t+1]$, $t = 0, 1, 2, \ldots$, for $\hat{r}[t+1]$ defined by (22), (23), and (24) (the AHC algorithm), there exists parameters α, β, and γ, such that*

$$\alpha\hat{r}[t+1] = \alpha'\hat{r}'[t+1], \qquad \forall t \geq 0$$

where $\hat{r}'[t+1]$ is defined by a similar "primed" set of equations:

$$p'^s[t] = \sum_{j=1}^{n} v'_j x_j[t],$$

$$v'_i[t+1] = v'_i[t] + \beta'\hat{r}'[t+1]\bar{x}_i[t], \qquad v'_i[0] = 0,$$

and

$$\hat{r}'[t+1] = a\, r[t+1] + b\, p'^t[t+1] - c\, p'^t[t],$$

for any constants α', β', a, b, c, with $\beta' > 0$, and $a, c \neq 0$.

Proof: We have that

$$\alpha'\hat{r}'[t+1] = \alpha'\left(a\, r[t+1] + b\, p'^t[t+1] - c\, p'^t[t]\right).$$

Choosing $\alpha = \alpha' a$ and $\gamma = \frac{b}{c}$,

$$\alpha'\hat{r}'[t+1] = \alpha\left(r[t+1] + \gamma\frac{c}{a}p''[t+1] - \frac{c}{a}p''[t]\right)$$
$$= \alpha\left(r[t+1] + \gamma\sum_{j=1}^{n}\frac{c}{a}v_j'[t]x_j[t+1] - \sum_{j=1}^{n}\frac{c}{a}v_j'[t]x_j[t]\right).$$

Since

$$\alpha\hat{r}[t+1] = \alpha\left(r[t+1] + \gamma\sum_{j=1}^{n}v_j[t]x_j[t+1] - \sum_{j=1}^{n}v_j[t]x_j[t]\right),$$

the theorem will be proven if it is shown that β can be chosen such that $v_i[t] = \frac{c}{a}v_i'[t], \forall t \geq 0, \ \forall i, \ 1 \leq i \leq n$. This is shown by induction, using $\beta = c\,\beta'$.

$$v_i[0] = \frac{c}{a}v_i'[0] = 0, \qquad \forall i, \quad 1 \leq i \leq n.$$

Combining (22), (23), and (24) gives

$$v_i[t+1] = v_i[t] + \beta\left(r[t+1] + \gamma\sum_{j=1}^{n}v_j[t]x_j[t+1] - \sum_{j=1}^{n}v_j[t]x_j[t]\right)\bar{x}_i[t].$$

By the inductive assumption and the choices of β and γ,

$$= \frac{c}{a}v_i'[t] + c\,\beta'\left(r[t+1] + \frac{b}{c}\sum_{j=1}^{n}\frac{c}{a}v_j'[t]x_j[t+1] - \sum_{j=1}^{n}\frac{c}{a}v_j'[t]x_j[t]\right)\bar{x}_i[t].$$

Rearranging terms,

$$= \frac{c}{a}\left[v_i'[t] + \beta'\left(a\,r[t+1] + b\,p''[t+1] - c\,p''[t]\right)\bar{x}_i[t]\right]$$
$$= \frac{c}{a}\left[v_i'[t] + \beta'\hat{r}'[t+1]\bar{x}_i[t]\right]$$
$$= \frac{c}{a}v_i'[t+1].$$

Q.E.D.

Conclusions

The ideal reinforcement signal is that which provides the highest quality evaluation of a learning system's behavior. Temporal credit-assignment algorithms can be viewed as attempts to approximate the ideal reinforcement signal. This theoretical construct assists in the understanding of existing temporal credit-assignment algorithms and the creation of new ones. It is possible to establish relationships between the ideal reinforcement signal and temporal credit-assignment algorithms that approximate it. In this chapter the ideal reinforcement signal has been used as the basis for an informal derivation of the AHC temporal credit-assignment algorithm. In such derivations there are several choice points from which different paths lead to different algorithms.

The AHC algorithm is closely related to the "learning-by-generalization" algorithm used in Samuel's checker-playing program (Samuel, 1959). If Samuel's algorithm is simplified in several ways, including the removal of all features specialized for the game of checkers, for tasks in which off-line search is performed, and for efficient implementation on a small computer, the only difference between what remains and the AHC algorithm is that the latter includes an additional parameter, the discount rate parameter. If the AHC algorithm's discount rate parameter is 1, it is identical to the simplified version of Samuel's algorithm. Theoretical analysis in terms of the ideal reinforcement signal suggests that while a non-unit discount rate parameter is not needed for the special class of time-blind tasks, which includes but is largely limited to game-playing tasks, it may be necessary for other tasks.

CHAPTER VI

EXPERIMENTS INVOLVING SECONDARY REINFORCEMENT

The experiments described in preceding chapters concern learning tasks in which stimuli are generated by an autonomous process unaffected by the behavior of the learning system. Henceforth such tasks are called ***autonomous-stimuli*** tasks. Tasks in which actions selected by the learning system are allowed to influence its subsequent stimuli are called ***nonautonomous-stimuli*** tasks. Removing the restriction to autonomous stimuli makes a surprisingly large difference in the effectiveness of some learning algorithms. The best-performing algorithms in the experiments of preceding chapters with autonomous-stimuli tasks are shown in this chapter to be among the worst-performing on nonautonomous-stimuli tasks.

Nonautonomous-stimuli tasks were avoided in preceding chapters because they provide opportunity for employing *secondary reinforcement*, that is, for using information provided by the occurrence of stimuli to perform temporal credit-assignment (see Chapter I). As long as stimuli are generated autonomously, they provide no information relevant to the evaluation of the learning system's behavior. In preceding chapters it was useful to exclude secondary reinforcement in order to investigate other issues in as simple a context as possible. Armed with the knowledge gained from studying these more restricted classes of tasks, in this chapter we consider tasks involving secondary reinforcement.

The first section of this chapter presents a series of simulations illustrating the

behavior of the AHC algorithm (see Chapter V) when repetitively presented with specific temporal patterns of stimuli and primary reinforcement. Following sections describe experiments with nonautonomous-stimuli tasks, three of which involve abstract finite-state environments, and one which involves a realistic pole-balancing environment. The secondary-reinforcement algorithms discussed in Chapter V and a variety of algorithms that do not implement secondary reinforcement are compared in performance on each task.

Illustrations of the Behavior of the AHC Algorithm

In this section the behavior of the AHC algorithm is illustrated by repeatedly presenting it with specific sequences of stimuli and primary reinforcement. Since the behavior of the AHC algorithm depends only on this input, all of which is autonomously generated, a full learning system need not be simulated. In these simulations, those parts of a full learning system that use the heuristic reinforcement signal produced by the AHC algorithm, which select actions and update the stimulus-action map, are omitted. In addition, since the operation of the algorithm is deterministic, multiple runs and tests of statistical significance are also unnecessary.

These simulations illustrate the behavior and operation of the AHC algorithm rather demonstrate its capabilities. In most cases, the AHC algorithm's discount-rate parameter γ is 1, so the behavior illustrated is also that of the SS algorithm (see Chapter V). In a few cases the behavior of the AHC algorithm is contrasted with that of a reinforcement-comparison algorithm; the intent in these is not to demonstrate a superior ability of the AHC algorithm, but only to illustrate by comparison certain problems with the reinforcement-comparison algorithm.

Figure 34 shows the behavior of the AHC algorithm (defined by (22), (23),

(24), and (15)) and of a reinforcement-comparison (RC) algorithm (defined by (17), (18), (20), and (15)) when repetitively presented with a brief stimulus followed by a brief period of primary reinforcement. Time trajectories of the input variables are shown in the upper part of the figure, and synchronized plots of $\hat{r}$ for the two algorithms on subsequent trials (in increments of 10) are shown below. A *trial* here is defined as a single complete presentation of the input sequence shown at the top of the figure. The first plot of $\hat{r}$ is due to a fictitious "Trial 0" in which the input sequence is presented to the algorithm and $\hat{r}$ is measured, but during which no learning occurs. This plot thus represents the initial behavior of each algorithm.

This simulation compares the behavior of the AHC and RC algorithms in one of the simplest cases in which stimuli provide information about forthcoming primary reinforcement. The stimulus is a "vector" of a single component. It is nonzero for only one time step, and is immediately followed by positive primary reinforcement, also lasting one time step. During an input event the corresponding input variable ($x_i[t]$ or $r[t]$) takes on the value 1; otherwise it is 0. The temporal relationships in this simulation have been chosen to illustrate in as pure and simple a form as possible both 1) the basic idea of a secondary-reinforcement algorithm, and 2) that the RC algorithm fails to capture this idea and thereby ends up being counterproductive.

The behavior of the AHC algorithm shown in Figure 34 is the ideal behavior for a realizable temporal credit-assignment algorithm in this situation. This ideal behavior includes characteristics of both secondary-reinforcement and reinforcement-comparison mechanisms. Reinforcement always follows the stimulus, so credit should be assigned ($\hat{r}$ should be positive) at the time of occurrence of the stimulus. This behavior—using a stimulus predictive of reinforcement to assign credit earlier—is the hallmark of a secondary-reinforcement algorithm. Since reinforcement is always preceded by the stimulus, no credit should be assigned ($\hat{r}$ should be 0) at the time of reinforcement, because the reinforcement has already been pre-

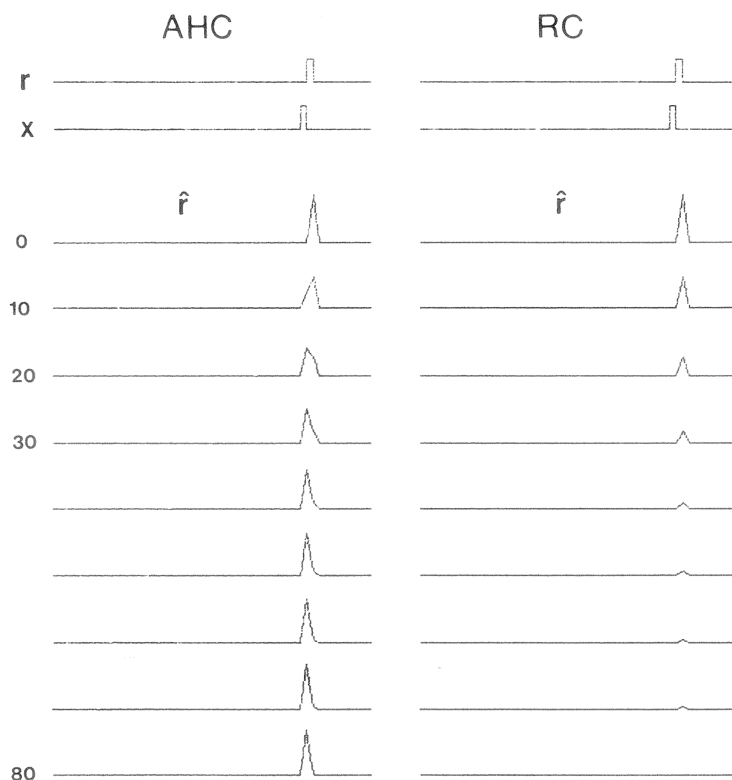


Figure 34. Behavior of the AHC and RC Algorithms when Presented with a Brief Stimulus Followed by Primary Reinforcement. Behavior on the left is due to the AHC algorithm, and behavior on the right is due to the RC algorithm. The upper traces show the input sequence repetitively presented to each algorithm. Below are synchronized plots of the behavior of $\hat{r}[t+1]$ on subsequent trials.

dicted. This second behavior is a consequence of the use of the difference between actual and predicted reinforcement to assign credit (to determine $\hat{r}$). It is characteristic of a reinforcement-comparison algorithm. The AHC algorithm produces both behaviors discussed above, illustrating that it implements both secondary-reinforcement *and* reinforcement-comparison mechanisms. The RC algorithm, on

the other hand, implements only a reinforcement-comparison mechanism, and thus produces only the first behavior: It eliminates the original reinforcement without creating earlier reinforcement (see Figure 34).

The behavior shown in Figure 34 is produced using the parameter values $\beta = .5$ and $\gamma = 1$. The trace decay parameter λ used in computing $\bar{x}_i$ via (15) is such that the time constant τ of the trace's exponential decay is 10 time steps. The corresponding value for λ is

$$\lambda = 1 - e^{-1/\tau}.$$

In this and all following illustrative simulations the trace variables of the algorithms (the $\bar{x}_i$) are set to zero between trials to simulate the effect of very long inter-trial intervals.

Figure 35 shows the behavior of the AHC algorithm when presented with a temporal sequence of four clearly distinguishable stimuli followed by a brief period of primary reinforcement. Time trajectories of the input variables are shown twice in the upper part of the figure, and synchronized plots of $\hat{r}[t+1]$ and $p^t[t+1]$ on successive trials from Trial 1 to Trial 10 appear below. This experiment used parameter values β=.6, $\gamma = 1$, and $\tau = 5$. The four stimuli are each 10 time steps in duration, and the reinforcement lasts 1 time step. The simulation does not quite reach its asymptotic state during the 10 trials shown here. Eventually, $\hat{r}$ is positive only at the onset of the most earliest stimulus, and $p^t[t+1]$ is positive and unchanging over the time interval during which stimuli are present.

Notice how the times of positive $\hat{r}$ gradually move earlier in the input sequence. The stimuli occurring temporally closest to reinforcement become associated with reinforcement first, and they then act as the basis for association of earlier stimuli. The behavior of p follows a similar pattern: p is first positive only just before reinforcement, then gradually becomes positive earlier to the extent allowed by the stimuli.

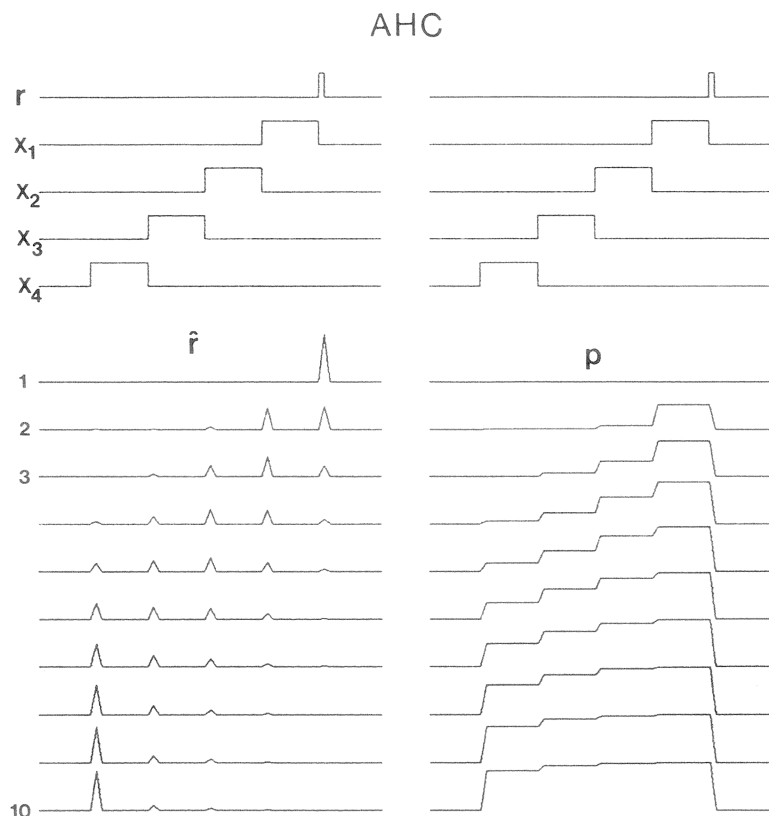


Figure 35. Behavior of the AHC Algorithm when Presented with a Temporal Sequence of Stimuli Followed by Primary Reinforcement. The upper traces show the input sequence repetitively presented to the AHC algorithm. Below are synchronized plots of the behavior of $\hat{r}[t+1]$ and $p^t[t+1]$ on successive trials.

Note that each positive $\hat{r}$ event, except for that at the time of reinforcement, matches in size and timing a corresponding increment in p. The size of the $\hat{r}$ event at the time of reinforcement is determined by the difference between the original effect of r on $\hat{r}$, shown in the Trial 1 plot, and the final decrement in p. In the last few trials shown, these exactly cancel out. These observations are readily related

to the equation defining $\hat{r}$ (Equation 24):

$$\hat{r}[t+1] = r[t+1] + \gamma\, p^t[t+1] - p^t[t].$$

During the time when reinforcement is not present, r is zero, leaving $\hat{r}$ determined only by the last two terms, which are roughly the change in p from one time step to the next. When reinforcement is present, its direct effect on $\hat{r}$ adds to a negative effect due to the decrement in p, leaving $\hat{r}$ determined by the difference in size of these two effects.

For comparison, Figure 36 shows the result of a simulation identical to the preceding one, except using the RC algorithm. With this algorithm an $\hat{r}$ event, if predictable, causes an $\hat{r}$ event of the opposite sign to precede it. The result, in this case, is that $\hat{r}$ and p exhibit oscillatory behavior. It seems unlikely that this behavior has any use in terms of temporal credit assignment.

Figure 37 summarizes the behavior of the AHC algorithm when presented with a stimulus and a brief period of primary reinforcement in 6 temporal configurations. The 6 input sequences are shown in the upper part of each segment of the figure. The lower part shows plots of the initial and near asymptotic behavior of $\hat{r}$ when the AHC algorithm is repetitively presented with the input sequence. In all cases the values of relevant parameter are $\beta = .02$, $\tau = 10$, and $\gamma = 1$. The near asymptotic behavior shown is actually the behavior on the 1000th trial.

In the simulations whose results are shown in Figures 37a and 37b, the offset of the stimulus precedes reinforcement by 20 and 10 time steps respectively. In both cases the stimulus becomes associated with reinforcement such that its onset causes $\hat{r}$ to become positive (indicating credit) and its offset causes $\hat{r}$ to become negative (indicating blame). In effect, the stimulus has come to be seen as a harbinger of high forthcoming reinforcement: Its onset is viewed as a good sign and its offset as a bad one. As a consequence, the occurrence and maintenance

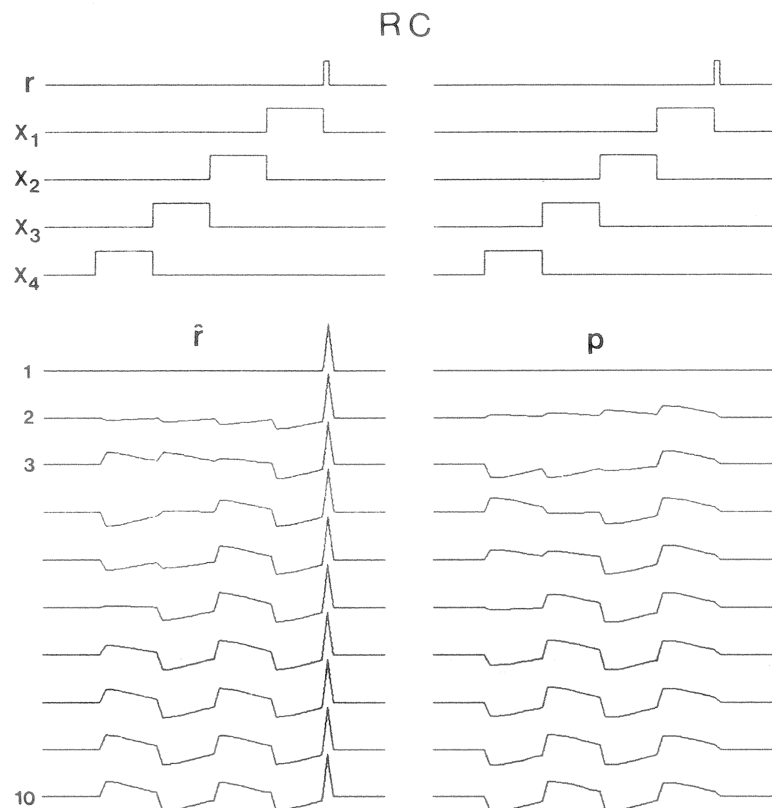


Figure 36. Behavior of the RC Algorithm when Presented with a Temporal Sequence of Stimuli Followed by Primary Reinforcement. The upper traces show the input sequence repetitively presented to the RC algorithm. Below are synchronized plots of the behavior of $\hat{r}[t+1]$ and $p^t[t+1]$ on successive trials.

of the stimulus would become desired and sought by the learning system. Note that the asymptotic association between the stimulus and reinforcement is stronger for the case shown in Figure 37b, in which the stimulus occurs in closer temporal proximity to reinforcement. Although these simulations alone are not enough to demonstrate it, association in fact falls off in an exponential fashion as the interval between stimulus offset and reinforcement onset increases, with the time constant of the exponential fall equal to τ. The level of association is maximal when this

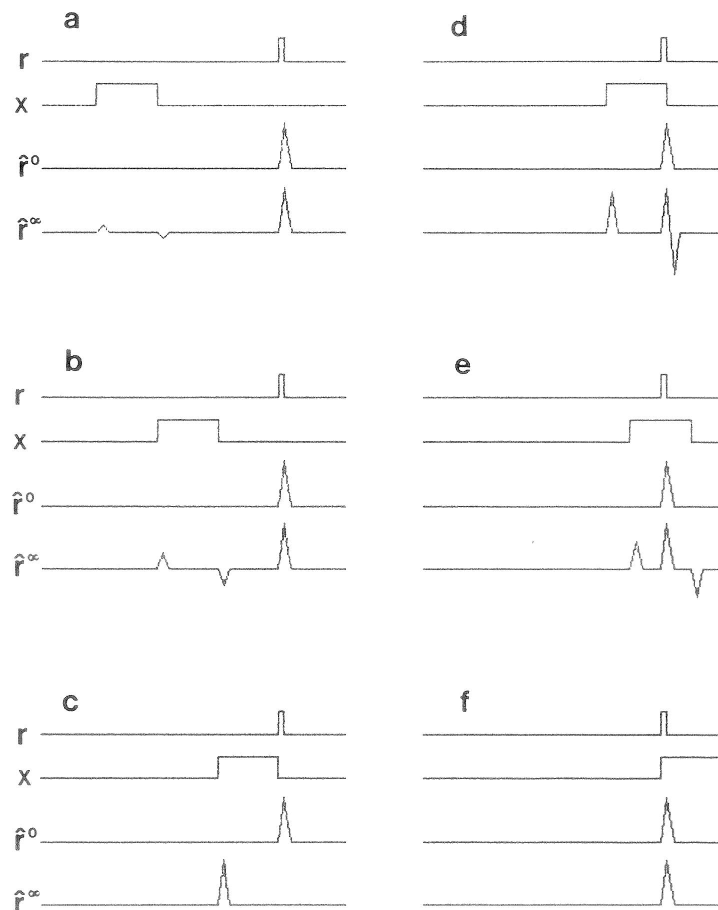


Figure 37. Behavior of the AHC Algorithm when Presented with a Stimulus and Primary Reinforcement in a Variety of Temporal Configurations. The upper traces of each segment of the figure show the input sequence repetitively presented to the algorithm. Below are synchronized plots of the initial and near asymptotic behavior of $\hat{r}[t+1]$.

time interval is zero, as in Figure 37c. In this case the negative effect on $\hat{r}$ due to the end of the stimulus coincides with and exactly cancels out the positive effect on $\hat{r}$ due to primary reinforcement.

If the offset of the stimulus occurs after the onset of reinforcement, as in the simulations shown in Figures 37d, 37e, and 37f, the asymptotic level of association of the stimulus with reinforcement decreases. If the onset of the stimulus occurs at the same time as (as shown in Figure 37f), or later than, the time of primary reinforcement, then no association is made.

Figure 38 summarizes the behavior of the AHC algorithm when presented with two stimuli and a brief period of primary reinforcement in 6 temporal configurations. As in Figure 37, the upper part of each segment of Figure 38 shows the 6 input sequences, and the lower part shows the initial and near asymptotic behavior of $\hat{r}$ when the indicated input sequence is repetitively presented to the AHC algorithm. In all 6 simulations the first stimulus, x_1, lasts 10 time steps and ends when reinforcement begins. The second stimulus, x_2, varies in its timing and duration over the 6 simulations, generally occurring later in subsequent experiments. The parameters values used are the same as those used in the preceding set of simulations.

In the simulation whose result is shown in Figure 38a, the first stimulus occurs just prior to reinforcement and becomes fully associated with reinforcement. The second stimulus, which precedes the first *stimulus* by 20 time steps, becomes associated with reinforcement to a lesser extent. This extent is approximately the same as that of the stimulus in Figure 37b, which precedes *reinforcement* by 20 time steps. In other words, the onset of x_1, the later stimulus, in Figure 38a is acting to create earlier associations just as primary reinforcement does in Figure 37b. The same phenomenon is seen in Figure 38b. Here x_2 ends when x_1 begins, and it becomes fully associated with reinforcement, just as the stimulus x that ends when reinforcement begins in Figure 37c becomes fully associated. Figure 38c is also related to Figure 37e is this way.

Figures 38d, 38e, and 38f illustrate competition between stimuli for association

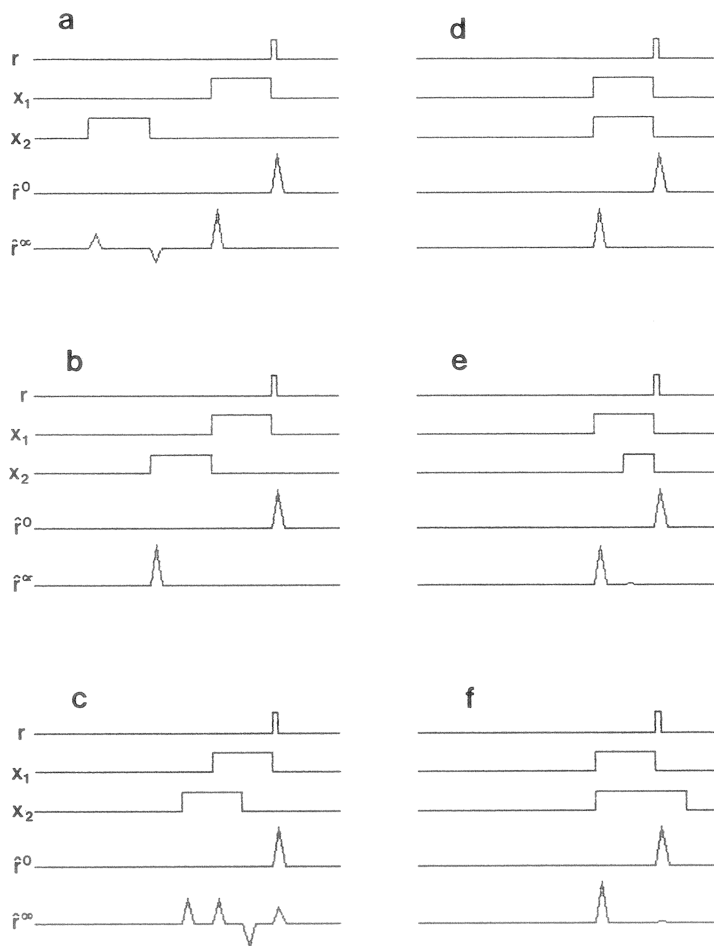


Figure 38. Behavior of the AHC Algorithm when Presented with two Stimuli and Primary Reinforcement in a Variety of Temporal Configurations. The upper traces of each segment of the figure show the input sequence repetitively presented to the algorithm. Below are synchronized plots of the initial and near asymptotic behavior of $\hat{r}[t+1]$.

with reinforcement. In Figure 38d the two stimuli are identically timed and each acquires a half strength association with reinforcement.* Note that the behavior of

* Unlike the other observations made in this section, this is not deducible from the plots shown, but has been determined by direct inspection of the reinforcement-association vector.

$\hat{r}$ here is the same as if only one stimulus had preceded reinforcement (as in Figure 37c). The association with reinforcement is shared equally between the two stimuli. For the cases shown in Figures 38e and 38f, on the other hand, no sharing occurs. In these two cases one of the two stimuli becomes fully associated with reinforcement and the other remains completely unassociated. This result is striking because the unassociated stimulus would have acquired substantial association if it had occurred alone. In these two cases one stimulus absorbs all the available association, including what would have gone, in its absence, to the other stimulus. For the case shown in Figure 38e, the stimulus that begins earlier is the one that becomes fully associated. For the case shown in Figure 38f, the stimulus that ends just as primary reinforcement begins is the one that becomes fully associated. In both cases the behavior exhibited moves the time at which $\hat{r}$ becomes positive as early as possible given the stimulus pattern, while minimizing other fluctuations in $\hat{r}$.

All preceding illustrations of the AHC algorithm's behavior have used the value 1 for the discount rate parameter γ. With $\gamma = 1$, the AHC algorithm is equivalent to the SS algorithm (see Chapter V). The simulation whose result is shown in Figure 39 illustrates how the AHC algorithm's behavior changes when γ is chosen less than 1. Figure 39 shows the initial and asymptotic behavior of $\hat{r}[t+1]$, and the asymptotic behavior of $p^t[t+1]$, for a simulation identical to that shown in Figure 35, with the exception that here $\gamma = .98$ instead of $\gamma = 1$. The largest positive $\hat{r}$ event in the asymptotic behavior with both $\gamma = 1$ and $\gamma = .98$ is at the onset of the earliest stimulus (see Figures 35 and 39). However, with $\gamma = 1$ there remain positive $\hat{r}$ events at the onset of each of the following stimuli, whereas with $\gamma = 1$, $\hat{r}$ at the asymptote is zero at all times other than the onset of the first stimulus.

For the AHC algorithm with $\gamma < 1$, stimuli that occur in closer temporal proximity to reinforcement are more highly valued. The arrival of each successive stimulus indicates that the forthcoming reinforcement is a little nearer, causing $\hat{r}$

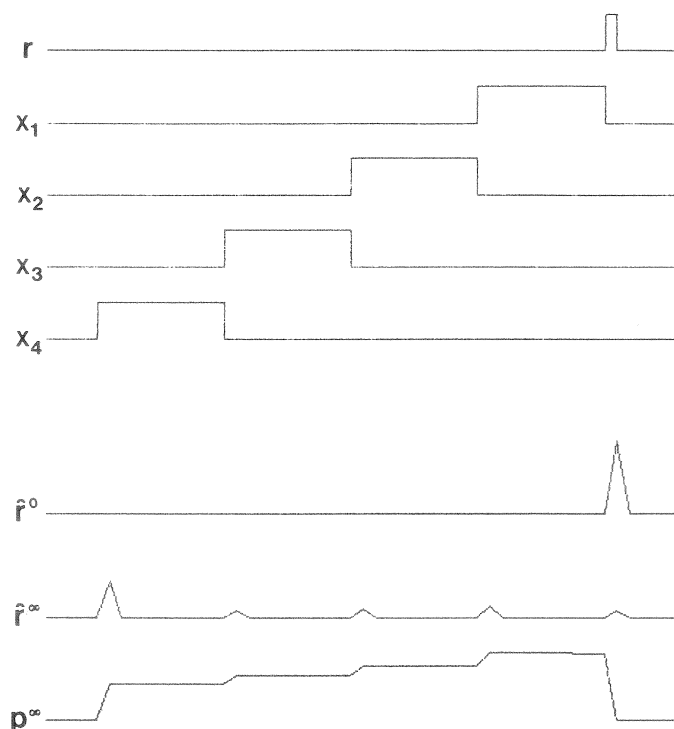


Figure 39. Behavior of the AHC Algorithm with $\gamma = .98$ when Presented with a Sequence of Stimuli Followed by Primary Reinforcement. The upper traces show the input sequence repetitively presented to the AHC algorithm. Below are plots of the initial and near asymptotic behavior of $\hat{r}[t+1]$, and the near asymptotic behavior of p^t.

to be positive and credit to be assigned. With $\gamma < 1$, the prediction p of reinforcement must increase from step to step in order to be sustained. This is why, in Figure 39, p *at asymptote* continues to increase during the input sequence. To sustain a given level of prediction it must be followed either by primary reinforcement or by a new higher level of prediction. If $\gamma = 1$, on the other hand, a prediction can be sustained indefinitely at a constant level. This qualitative consequence of the small quantitative change from $\gamma = 1$ to $\gamma < 1$ may be the cause of the large

changes in performance that were observed when this change is made in some of the experiments discussed later in this chapter.

Sutton and Barto's Classical Conditioning Model

Sutton and Barto presented an adaptive element analog of classical conditioning that is closely related to the AHC algorithm (Sutton and Barto, 1981; Barto and Sutton, 1982). They presented many simulations where the results, properly interpreted, apply to the AHC algorithm as well. This section details the relationship between the two algorithms and shows how to interpret the earlier results obtained with the classical conditioning model as further illustrations of the behavior of the AHC algorithm.

There are two differences between the notation used here and that used in earlier papers to describe the classical conditioning model. The first difference involves the time at which stimuli are said to occur. What is denoted in earlier papers concerning the classical conditioning model as $x_i[t-1]$ is denoted here as $x_i[t]$. The second difference involves the timing of the trace variables. What are denoted in earlier papers describing the classical conditioning model by $\bar{y}[t-1]$ and $\bar{x}[t-1]$ are here denoted by $\bar{y}[t]$ and $\bar{x}[t]$ (defined by (15)). Making these changes in notation, the main equation defining the classical conditioning model can be written as

$$v_i[t+1] = v_i[t] + \beta\,(y[t] - \bar{y}[t-1])\,\bar{x}_i[t], \tag{41}$$

where

$$y[t] = \sum_{i=0}^{n} v_i[t]x_i[t+1]. \tag{42}$$

All published simulation experiments with the classical conditioning model used

a degenerate form of the trace $\bar{y}[t-1]$ that is equivalent to $y[t-1]$. Assuming this case, (41) and (42) can be combined and rewritten as

$$v_i[t+1] = v_i[t] + \beta \left(\sum_{j=0}^{n} v_j[t]x_j[t+1] - \sum_{j=0}^{n} v_j[t-1]x_j[t] \right) \bar{x}_i[t].$$

In the classical conditioning model, input pathway x_0 corresponds to the unconditioned stimulus (UCS) pathway. The corresponding weight, v_0, is not modifiable. This input plays a role similar to that of the piece-advantage term in Samuel's checker-playing program (see Chapter V) and to that of primary reinforcement in a reinforcement learning task. The change in x_0 from time step to time step is what corresponds most directly to primary reinforcement in the classical conditioning model: $r[t+1] = v_0 x_0[t+1] - v_0 x_0[t]$. Making this substitution allows us to write the classical conditioning model as

$$v_i[t+1] = v_i[t] + \beta \left(r[t+1] + \sum_{j=1}^{n} v_j[t]x_j[t+1] - \sum_{j=1}^{n} v_j[t-1]x_j[t] \right) \bar{x}_i[t].$$

Converting to the $p^s[t]$ notation defined by (22) yields

$$v_i[t+1] = v_i[t] + \beta \left(r[t+1] + p^t[t+1] - p^{t-1}[t] \right) \bar{x}_i[t]. \tag{43}$$

The algorithm given by (43) is equivalent to the classical conditioning model studied by Sutton and Barto. It is also almost identical to the AHC algorithm with $\gamma = 1$, i.e., to the SS algorithm (cf. (23) and (24)). The only difference is that the classical conditioning algorithm uses $p^{t-1}[t]$ in its subtractive p term while the AHC algorithm uses $p^t[t]$, i.e., the classical conditioning algorithm uses $\vec{v}[t-1]$ here while the AHC algorithm uses $\vec{v}[t]$. Since $\vec{v}$ is expected to change very little from one time step to the next, this difference should have little practical effect under normal conditions. The results obtained with the classical conditioned model should also hold without significant change for the AHC algorithm with $\gamma = 1$,

where the change in the UCS signal (x_0) from time step to time step should be viewed as the primary reinforcement signal (as described above).

The classical conditioning model, the simplified version of Samuel's algorithm, and the AHC algorithm with $\gamma = 1$ are all nearly identical. It is intriguing that such disparate goals as making a high-performance checker-playing program, accurately modeling certain classical conditioning data, and approximating formalizations of an ideal reinforcement signal can yield such similar algorithms.

Experiment 1: Reinforcement Anticipation

The state-transition structure of the environment used in Experiment 1 is shown in Figure 40. The environment has 4 states, labeled 1 through 4, organized in a loop. Primary reinforcement is zero at all times except when the loop is completed and the environment returns to its initial state. At each time step the environment either remains in the same state or moves one state farther along the loop. The probability with which these two alternatives occur depends on the action selected. If Action 1 is selected, the environment moves to its next state with probability .6 and remains in its current state with probability .4. These two probabilities are reversed if Action 0 is selected. (Action 1 corresponds to the right-side actions in Figure 40.) Since reinforcement is maximized by progressing around the loop as rapidly as possible, Action 1 is the better (correct) action from all states.

The stimulus representation of states on this task is such that no state is similar to any other. Corresponding to the i th state is a stimulus vector $\vec{x}^i$ all of whose components are zeros except for the i th, which is 1 (e.g., $\vec{x}^2 = (0,1,0,0)$).

Table 7 lists the 10 algorithms applied to the task. Algorithms 4, 5, 8, and 9 are

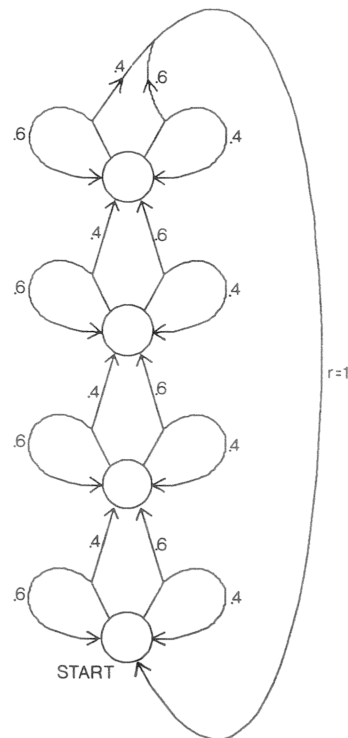


Figure 40. State-Transition Structure of the Environment used in Experiment 1 of Chapter VI. Each circle represents a state of the environment. State transitions are probabilistically dependent on the action selected by the learning system. The labels on the arc branches indicate the probability with which that transition occurs given that the arc's action is selected. Left arcs correspond to Action 0, right arcs to Action 1. The expected value of the reinforcement signal is 0 for all state transitions except that from the top state back to the start state.

identical to the like-numbered algorithms of Chapter IV. Algorithms 12 and 13 use the AHC algorithm. Algorithm 12 uses an eligibility factor based on $y[t] - \frac{1}{2}$, and Algorithm 13 uses an eligibility factor based on $y[t] - \pi[t]$. Similarly, Algorithms 14 and 15 use the SS algorithm (defined by (22), (23), and (38)), and Algorithms 16 and 17 use Witten's algorithm (defined by (22), (23), and (40). All algorithms

Table 7. Learning Algorithms of Chapter VI.

Number	Update Rule
4,5	$w_i[t+1] = w_i[t] + \alpha r[t+1]\bar{e}_i[t]$
8,9	$w_i[t+1] = w_i[t] + \alpha(r[t+1] - p^t[t])\bar{e}_i[t]$
	$v_i[t+1] = v_i[t] + \beta(r[t+1] - p^t[t])\bar{x}_i[t]$
12,13	$w_i[t+1] = w_i[t] + \alpha(r[t+1] + \gamma p^t[t+1] - p^t[t])\bar{e}_i[t]$
	$v_i[t+1] = v_i[t] + \beta(r[t+1] + \gamma p^t[t+1] - p^t[t])\bar{x}_i[t]$
14,15	$w_i[t+1] = w_i[t] + \alpha(r[t+1] + p^t[t+1] - p^t[t])\bar{e}_i[t]$
	$v_i[t+1] = v_i[t] + \beta(r[t+1] + p^t[t+1] - p^t[t])\bar{x}_i[t]$
16,17	$w_i[t+1] = w_i[t] + \alpha((1-\gamma)r[t+1] + \gamma p^t[t+1])\bar{e}_i[t]$
	$v_i[t+1] = v_i[t] + \beta((1-\gamma)r[t+1] + \gamma p^t[t+1])\bar{x}_i[t]$

Where:

$$e_i[t] = \begin{cases} (y[t] - 1/2)x_i[t], & \text{for even numbered algorithms;} \\ (y[t] - \pi[t])x_i[t], & \text{for odd numbered algorithms.} \end{cases}$$

$$w_i[0] = 0 \qquad v_i[0] = 0 \qquad y[t] \in \{1, 0\} \qquad \alpha > 0 \qquad \beta = .05 \qquad 0 < \gamma < 1$$

$$y[t] = \begin{cases} 1, & \text{if } s[t] + \eta[t] > 0; \\ 0, & \text{otherwise.} \end{cases}$$

where $\eta[t]$ is a normally distributed random variable of mean 0 and standard deviation $\sigma_y = .1$, and

$$s[t] = \sum_{i=1}^{n} w_i[t]x_i[t]$$

$\pi[t]$ is the probability that y[t]=1, given $\vec{x}[t]$

$$p^s[t] = \sum_{i=1}^{n} v_i[s]x_i[t] \qquad p[0] = r[1]$$

For any time sequence $z[t]$, $\bar{z}[t]$ is defined by

$$\bar{z}[t] = (1-\delta)\bar{z}[t-1] + \delta z[t] \qquad \bar{z}[0] = 0 \qquad 0 < \delta < 1$$

were run at a range of values for the α parameter, and at fixed values for the other parameters. The α values used were the powers of 2 from 2^{-8} to 2^{-2}. The other parameter values used were $\beta=.1$, $\delta=.5$, and $\gamma=.95$.

100 separate simulation runs were made with each algorithm, each run differing only in the initial seed of the random number generator. Runs were terminated after 300 complete trips had been made around the 4-state loop. At the end of each run the final probability of taking each action in each state was computed and recorded, as discussed in previous chapters. As a measure of performance on the run, the expected value of the time it would take to complete one circuit around the loop, with the final probabilities found by the algorithm, was computed as follows, where p_i denotes the final probability of taking Action 0 when the environment is in state i:

$$\sum_{i=1}^{4} \frac{1}{.4+.2p_i}.$$

Using this performance measure, the algorithms that performed best are those with the *lowest* score.

The expected circuit times were averaged over the 100 runs with each algorithm at each α value to yield the data shown in Figure 41. This figure plots the average performance levels due to each algorithm at each α value. The upper dashed line indicates the initial, chance performance level achieved when both actions are selected with equal probability from all states. The lower dashed line indicates the optimal performance level, that achieved when Action 1 is selected with probability 1 from all states.

The AHC algorithms (12 and 13) and the SS algorithms (14 and 15) performed best on this task. The AHC algorithms performed slightly but significantly better than the SS algorithms, and the algorithms with eligibility terms based on $y[t]-\pi[t]$ (13 and 15) performed slightly but significantly better than those with eligibility terms based on $y[t]-\frac{1}{2}$ (12 and 14). Algorithms 4 and 5, the algorithms based on

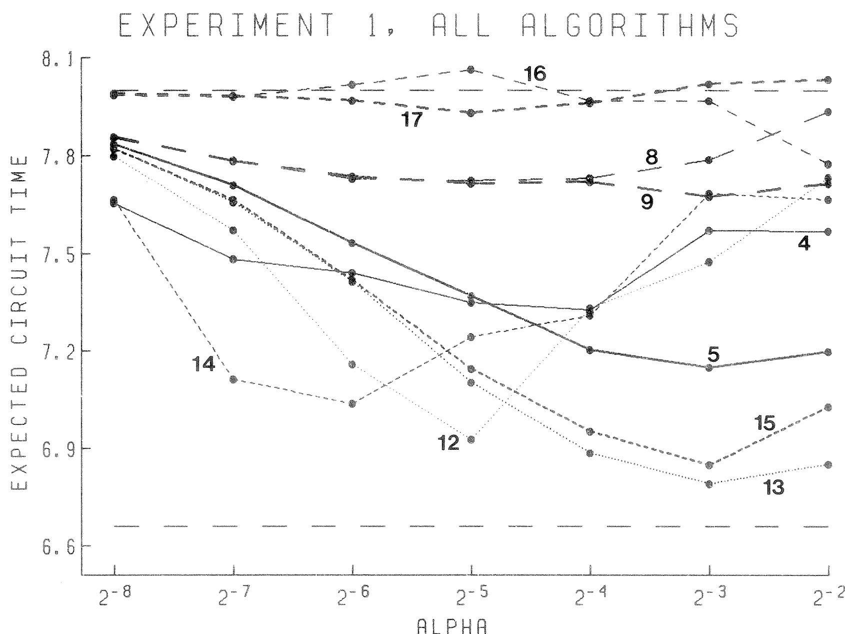


Figure 41. Algorithm Performance on Experiment 1 of Chapter VI. Low values correspond to better performances on this graph. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of the time it would take to complete a full trip around the environment shown in Figure 40. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The lower dashed line marks the optimal performance level, and the upper dashed line marks the chance performance level.

pure averaging without a heuristic-reinforcement component, performed next best, followed by Algorithms 8 and 9, and then by Algorithms 16 and 17.

Algorithm 5, with an eligibility term based on $y[t] - \pi[t]$, performed significantly better than Algorithm 4, which is identical except that its eligibility term is based on $y[t] - \frac{1}{2}$. For Algorithms 4 and 5 this task is a series of noisy delayed-reinforcement tasks, one corresponding to each state of the environment. The noise is due both to the environment making state transitions non-deterministically and

to the variation in the actions selected from later states. These algorithms learned slowly because delayed-reinforcement tasks, particularly noisy ones, are very difficult, as was seen in Experiment 1 of Chapter IV.

Based on the illustrative simulations presented earlier in this chapter, it is easy to understand why the RC algorithms (8 and 9) performed so poorly on this task, and why Algorithms 12-15 performed so well. Stimuli indicating the environment's state, particularly those indicating later states in the loop, can be used to predict the final reinforcement received on completing the loop. As we saw in the earlier simulations, the RC algorithm uses predictive information to "cancel out" nonzero reinforcement, eliminating its ability to reinforce. Algorithms 8 and 9 performed poorly because the RC algorithm eliminated the reinforcement they need in order to learn. The AHC and SS algorithms, on the other hand, work on this task much like they do in the simulation whose result is shown in Figure 35: Stimuli become associated with reinforcement according to their temporal proximity to it. As a consequence, $\hat{r}$ becomes positive each time a new state in the loop is reached. Because Algorithms 12-15 received this immediate rather than delayed reinforcement, they performed much better than the other algorithms.

The performance of Algorithms 16 and 17, based on Witten's temporal credit-assignment algorithm, was scarcely above the chance level. Whereas the performance of most algorithms was an inverted-U shaped function of α, with the optimal α clearly within the tested range, for Algorithms 16 and 17 it seems possible that performance might have been better at higher α values. However, additional simulation runs with higher α values, not reported in Figure 41, have ruled out this possibility. The performance of these algorithms becomes more erratic at higher α values, but not better. The poor performance of these algorithms that include a secondary-reinforcement mechanism but that do not include a reinforcement-comparison mechanism suggests that these two features should be combined, as they are in Algorithms 12-15, in order to take effective advantage of secondary

reinforcement on this task.

Many of the algorithms using $y[t] - \pi[t]$ in their eligibility factors performed slightly better than the corresponding algorithms using $y[t] - \frac{1}{2}$. The $y[t] - \frac{1}{2}$ algorithms also performed best at lower α values than those at which $y[t] - \pi[t]$ algorithms performed best. Probably, $y[t] - \frac{1}{2}$ algorithms perform worse because they settle too rapidly on the action choice that looks good early on and then never try the other. The following two experiments shed more light on this issue.

Experiment 2: An "Easy" Action Sequence

On delayed-reinforcement tasks, long eligibility traces are essential for algorithms (such as Algorithms 4, 5, 8, and 9) that do not attempt to anticipate the delivery of reinforcement via secondary reinforcement. The need to perform a sequence of actions to get reward is a common way for delayed reinforcement to arise. The task used in Experiment 2 is one of the simplest of such cases, and one which one might think would be very easy. The experiment illustrates a serious problem for some of the less sophisticated algorithms that can occur when reinforcement is delayed and eligibility traces are long.

The task in this experiment is like that of Experiment 1 except that it is much easier. The environment is diagrammed in Figure 42. The 4 states are arranged in a loop, with positive reinforcement available only on the completion of a full circuit around the loop. This environment differs from that of Experiment 1 in that its behavior is a deterministic function of the learning system's action. An action of 1 always results in the environment moving ahead by one state in the loop of states, and an action of 0 always results in the environment remaining in the same state.

As in the environment of Experiment 1, all states are completely distinguishable

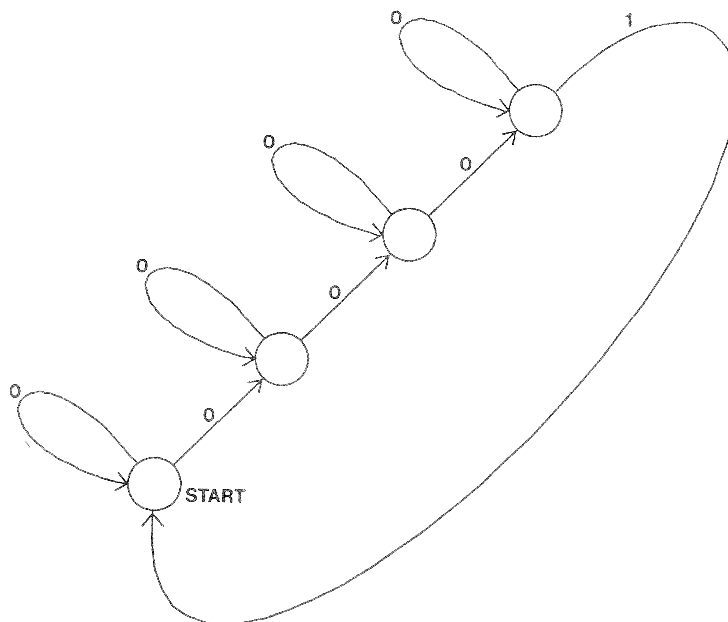


Figure 42. State-Transition Structure of the Environment used in Experiment 2 of Chapter VI. This simple environment causes surprising difficulties for some algorithms. Each circle represents a state of the environment. Arc labels indicate the expected value of primary reinforcement should that state transition occur. Which state transition occurs (which arc is followed) depends deterministically on the action selected by the learning system. All states are completely dissimilar.

and dissimilar. Corresponding to the ith state is a stimulus vector $\vec{x}^i$ all of whose components are zero except for the ith, which is 1 (e.g., $\vec{x}^2 = (0, 1, 0, 0)$).

All of this chapter's algorithms were applied to this task, each with a range of values for α, and at fixed values for the other parameters. The α values used were the powers of 2 from 2^{-8} to 2^3. For the purposes of this experiment, the eligibility traces used in updating $\vec{w}$ needed to be long, but the traces used in updating $\vec{v}$ did not. To accommodate two different traces, two different trace parameters, δ_w

and δ_v, were used. δ_w was chosen near zero (.1) so as to make the eligibility traces correspondingly long. The other parameter values were $\delta_v = .5$, $\beta = .3$, and $\gamma = .95$.

100 separate simulation runs were made with each algorithm, each run differing only in the initial seed for the random number generator. Runs were terminated after 100 complete trips had been made around the 4-state loop, or if more than 5000 steps passed without completing a trip. At the end of each run the final probability of taking each action in each state was computed and recorded, as discussed in previous chapters. The expected value of the time to complete one circuit around the loop, with the final probabilities found by the algorithm, was also computed at this time.

The expected circuit times were averaged over the 100 runs with each algorithm and α value. The inverse of the average expected circuit time was used as a measure of performance of each algorithm at each α value. Figure 43 plots these performance levels, with those due to the same algorithm at different values of α connected by lines. The lower dashed line indicates the initial, chance performance level achieved if both actions are selected with equal probability from all states. The upper dashed line indicates the optimal performance level achieved if the correct action (1) is selected with probability 1 from all states.

Performances varied widely on this task, both among algorithms and among different α values with the same algorithm. Algorithms 4, 14, and 16 performed near or slightly above the chance level for very low α values and at the zero level for higher α values. Algorithms 5, 13, and 15, on the other hand, performed near optimally at several α values, although they too performed at the zero level for the highest α values. Note that the zero performance level is significantly worse than the chance performance level. In these cases the algorithm learned to always (or nearly always) make the wrong choice of action from one of the 4 states, leaving

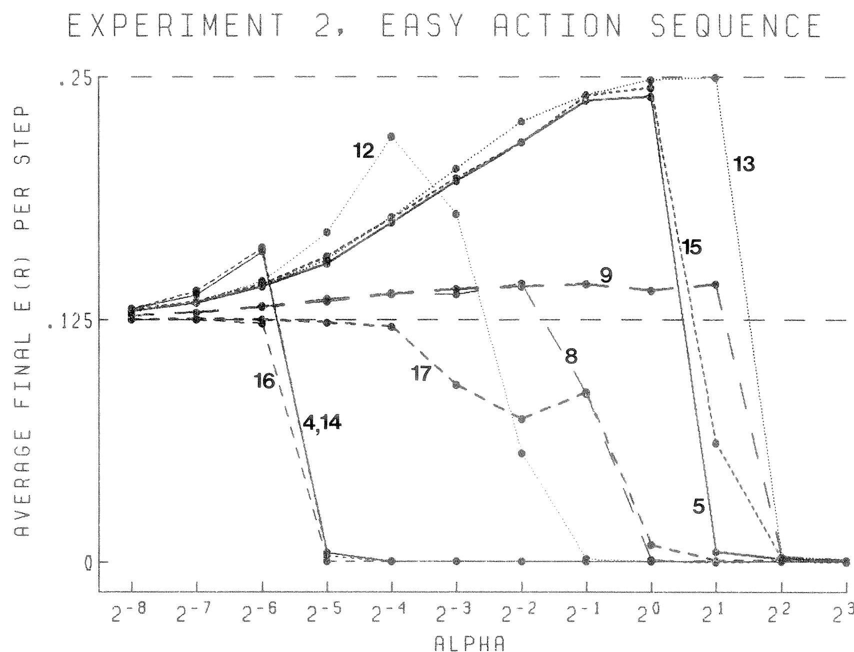


Figure 43. Algorithm Performance on Experiment 2 of Chapter VI. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement per time step at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The upper dashed line marks the optimal performance level, and the lower dashed line marks the chance performance level.

the learning system trapped in that state.

This task seems like a straightforward and even easy case in which a sequence of actions leads to reinforcement. Why do all algorithms "get stuck" at sufficiently high α values, and, more importantly, why do so many algorithms, e.g., Algorithms 4, 14, and 16, perform so poorly at all α values on this task? At root, the reason for the difficulty involves the long eligibility traces used and the resultant conflict between recency and frequency credit-assignment heuristics.

Early in a run the incorrect action is probably selected by chance several times from one of the states before the correct action is selected. When reinforcement is finally received, the incorrect action, even though it delayed the delivery of reinforcement, may well be assigned more credit than the correct one. The correct action was selected in closer temporal proximity to reinforcement than the incorrect action, but the incorrect action was selected more often. With long eligibility traces, the frequency heuristic may easily outweigh the recency heuristic and assign greater credit to the incorrect action. As a result, on the next trip around the loop, the learning system is more likely to choose the incorrect action, increasing the likelihood that it is selected multiple times before the correct action is finally selected. This in turn accentuates the tendency to assign credit incorrectly. The learning algorithm becomes caught in a vicious circle: Each time it manages to get past its "stuck state," its tendency to get stuck becomes stronger. The learning system ends up forever choosing the incorrect action and never receiving reinforcement.

Algorithms are least susceptible to this problem at low α values. The vicious circle is entered only after an algorithm gets a significant start towards selecting the wrong action; if learning is slow enough this is unlikely to happen.

The results shown in Figure 43 indicate that the even-numbered algorithms, those using $y[t] - \frac{1}{2}$ in their eligibility factors, are much more susceptible to this problem than the odd-numbered algorithms, which use $y[t] - \pi[t]$. In the case of the latter algorithms, when one action is more likely to be chosen than the other, its eligibility is discounted proportionally. This discounting exactly counteracts the extra credit the action tends to receive due to its greater frequency, and thus avoids the vicious circle.

Although the difference in eligibility terms explains most of the performance variations among algorithms on this task, there remain some clear effects due to the form of heuristic reinforcement used. The use of the AHC algorithm significantly

improves performance; Algorithm 12 performed much better than Algorithm 4, and Algorithm 13 performed slightly better over a wider range of α values than did Algorithm 5. The SS algorithms (14 and 15) performed nearly identically to the algorithms with no heuristic reinforcement mechanism (4 and 5), and worse than the AHC algorithms (12 and 13). The two algorithms using Witten's algorithm (16 and 17) never exceeded the chance performance level at any α value. Finally, the pure reinforcement-comparison algorithms (8 and 9) performed only slightly better than chance.

The difficulties that some algorithms have on this task are due to their long eligibility traces. Although the AHC algorithms perform better than those algorithms without a reinforcement-anticipation mechanism in this experiment, a more important advantage of such secondary-reinforcement algorithms in this regard is that they greatly reduce the need for such long traces. Algorithms 4 and 5 must have traces sufficiently long to span the full time from action to primary reinforcement. If reinforcement can be anticipated and moved temporally closer to the actions that cause it, then eligibility traces can be much shorter.

Experiment 3: Misleading Generalizations

Chapters III and IV examined several forms of misleading generalization and the ability of various algorithms to overcome them. Experiment 4 of Chapter IV, in particular, involved a case of misleading generalizations accentuated by different lengths of delays between action and reinforcement for two similar stimuli. Experiment 3 of this chapter concerns a similar situation, brought about when a sequence of actions is required in order to obtain reinforcement.

In tasks which require a correct sequence of actions, it is common for the correct action to be different at different points in the sequence. If any pair of such

points involves similar stimuli, then a particularly difficult problem of misleading generalization may arise. Figure 44 diagrams the environment used in Experiment 3; it is one of the simplest cases in which this problem may occur. In this task, a sequence of two actions is required for positive reinforcement. The two actions are different, and the corresponding stimulus vectors are similar according to the linear mapping metric of similarity (positive inner product). Upon completion of the correct action sequence, the environment returns to its initial state. From the first state, Action 0 is correct, since it leads on to the second state, whereas Action 1 leaves the environment in the first. Either action results in a primary reinforcement signal value of zero. From the second state, Action 1 is correct, since it results in primary reinforcement of +1 and the environment returning to the first state, whereas Action 0 results in zero primary reinforcement and the environment remaining in the second state. The stimulus vectors for the two states $\vec{x}^1$ and $\vec{x}^2$ are the same as those used in earlier misleading generalization experiments; they are also shown in Figure 44.

All algorithms listed in Table 7 were applied to this task, each with a range of values for α, and at fixed values for the other parameters. The α values used were the powers of 2 from 2^{-10} to 2^1. For the purposes of this experiment, the eligibility traces used in updating $\vec{w}$ needed to be short, but the traces used in updating $\vec{v}$ did not. Accordingly, $\delta_w = .9$ and $\delta_v = .5$. The other parameter values were $\beta = .3$, and $\gamma = .95$.

100 separate simulation runs were made with each algorithm, each run differing only in the initial seed of the random number generator. Runs were terminated after 50 complete trips had been made from the first state and back via the second state, or if more than 2000 steps passed without completing a trip. At the end of each run the final probability of taking each action in each state was computed and recorded, as discussed in previous chapters. The expected value of the time to correctly complete a trip, with the final probabilities found by the algorithm, was

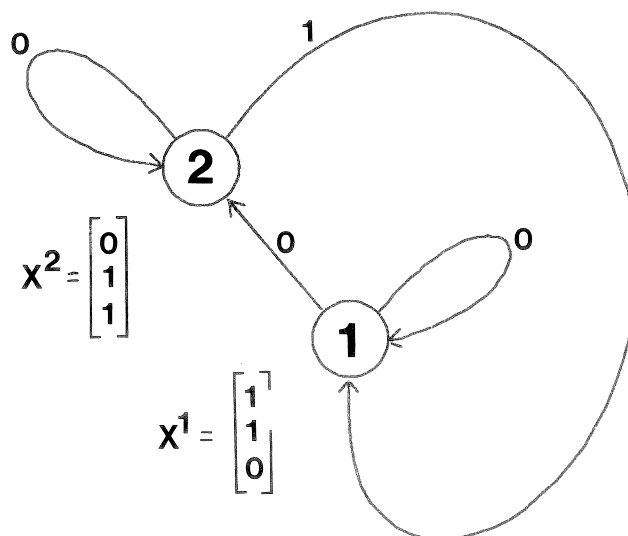


Figure 44. State-Transition Structure of the Environment used in Experiment 3 of Chapter VI. Each circle represents a state of the environment. Arc labels indicate the expected value of primary reinforcement should that state transition occur. Which state transition occurs (which arc is followed) depends deterministically on the action selected by the learning system. Left arcs correspond to Action 0, right arcs to Action 1. The similar stimulus vectors presented in each state are also shown.

also computed at this time.

The expected circuit times were averaged over the 100 runs with each algorithm and α value. The inverse of the average expected circuit time was used as a measure of performance of each algorithm at each α value. Figure 45 plots these performance levels, with those due to the same algorithm at different values of α connected by lines. The lower dashed line indicates the initial, chance performance level achieved if both actions are selected with equal probability from both states. The upper dashed line indicates the optimal performance level achieved if the

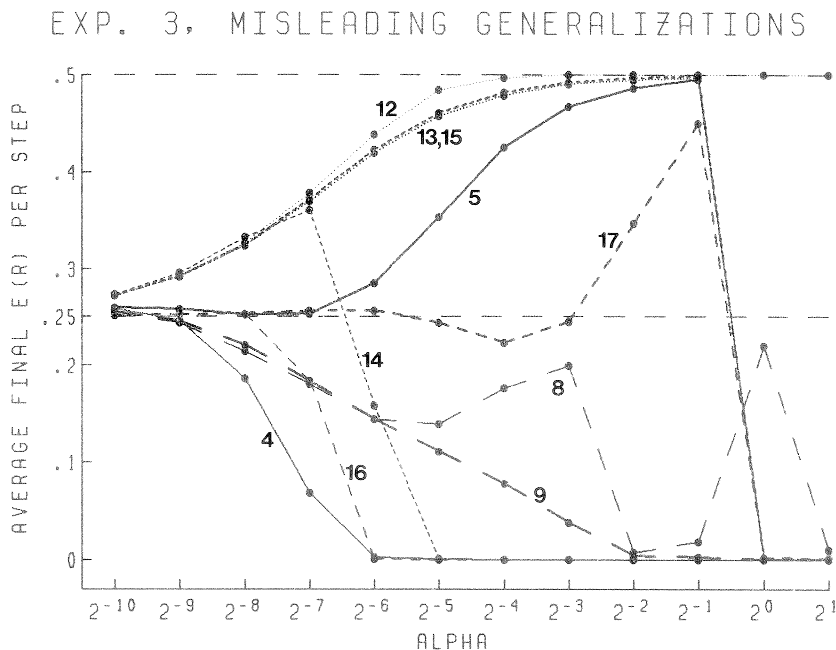


Figure 45. Algorithm Performance on Experiment 3 of Chapter VI. Each point represents the average performance over all runs with a particular algorithm and α value, where performance on a run is defined as the expected value of primary reinforcement per time step at the end of the run. Points due to the same algorithm (with different α values) are connected by lines; the numeric label indicates the associated algorithm. The upper dashed line marks the optimal performance level, and the lower dashed line marks the chance performance level.

correct action is selected with probability 1 from both states.

The results were similar to those of the preceding experiment. Performance varied widely, both among algorithms and among different α values with the same algorithm. All algorithms except one performed at the zero level with the highest α value. At lower values, the even-numbered algorithms performed very poorly, with the exception of Algorithm 12. Most of the odd-numbered algorithms performed near optimally over a range of α values.

An algorithm performing at the zero level on this task is performing much worse than the chance level. In all of these cases the learning system learned to make the incorrect action from the first state with a very high probability. This caused it to get stuck at that state and never complete a full trip. The reason for this is clear: Different actions are correct in each state, yet the stimuli are similar. The learning in both states generalizes inappropriately to the other. The learning in the second state proceeds more rapidly than that in the first because reinforcement is less delayed there. For some algorithms, the learning from the second state proceeds so rapidly that its generalization to the first overwhelms the learning taking place there and causes the learning system to always select the incorrect action. For these algorithms, the more learning takes place, the worse they perform.

As in the preceding experiment, the algorithms using $y[t]-\frac{1}{2}$ in their eligibility factors (the even-numbered algorithms) are much more susceptible to the problem than those using $y[t] - \pi[t]$ (the odd-numbered algorithms). Algorithm 4, for example, never significantly exceeded the chance performance level at any α value, while Algorithm 5, identical except for this difference in eligibility factors, attained near optimal performance at several α values. The best performing algorithms were 12, 13, 15, and 5. Apparently, the AHC algorithm and the SS algorithm improve performance on this task, because Algorithms 12 and 14 performed better than Algorithm 4, and Algorithms 13 and 15 performed near optimally over a wider range of α values than did Algorithm 5. Algorithm 12, in particular, performed best of all algorithms, yet it differs from Algorithm 4, which performed worst of all algorithms, only in that it uses the AHC algorithm.

The pure reinforcement-comparison algorithms (8 and 9) performed very poorly on this task, often falling below and never significantly exceeding the chance level. Of the algorithms using Witten's algorithm, the performance of Algorithm 16 often fell below and never significantly exceeded the chance level, and the performance of Algorithm 17 significantly exceeded the chance level only for a narrow range of

α values.

Experiment 4: Pole-Balancing*

The pole-balancing task is the most complex and realistic task on which the AHC algorithm has been tested. Figure 46 shows a schematic representation of the pole-balancing task. The rigid pole is hinged to a cart, which is free to move within the bounds of a 1-dimensional track. The pole is free to move only in the vertical plane of the cart and track. The learning system applies either a "left" or "right" force of fixed magnitude to the cart at each time step.

The cart-pole system has four state variables:

x: the position of the cart on the track,

θ: the angle of the pole with the vertical,

$\dot{x}$: the cart velocity, and

$\dot{\theta}$: the rate of change of the angle.

The pole begins upright ($\theta = 0$) and stationary ($\dot{\theta} = 0$), and the cart begins in the center of the track ($x = 0$) and stationary ($\dot{x} = 0$). If the pole falls over more than 12 degrees from vertical, or if the cart hits either edge of the track, a failure is said to occur. The object of the task is to prevent such failures for as long as possible. When a failure occurs, the cart-pole system is reset to its initial position, and a new attempt to balance the pole begins. A complete balancing attempt, from initial position to failure, is called a *trial*.

---

* Some of the results presented in this section have also been published by Barto, Sutton, and Anderson (in press). The simulations described in this section were programmed by Chuck Anderson.

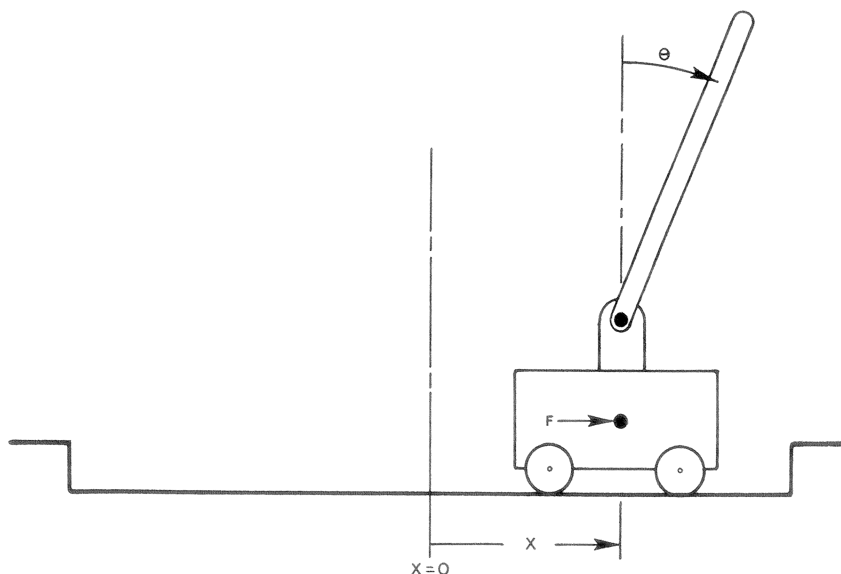


Figure 46. The Cart-Pole System to be Controlled in Experiment 4 of Chapter VI. The rigid pole is hinged to the cart, which is free to move within the bounds of the 1-dimensional track. The learning system attempts to keep the pole balanced and the cart within bounds by applying a force of fixed magnitude to the cart. (Figure by A. Barto)

No initial knowledge of the dynamics of the cart-pole system is assumed. The learning system initially knows only that it is to avoid failure. The credit-assignment problem in this task is a very difficult one. Since the actions that cause a failure may be taken many time steps before that failure actually occurs, this is a delayed-reinforcement task. Since failure is almost certain from some states of the cart-pole system, and nearly impossible (at least in the short run) from others, this is also an unbalanced-reinforcement task involving reinforcement-level asymmetry (see Chapter III).

The primary reinforcement signal r for this task is zero at all times except on failure. If the action selected at time t causes $\theta[t+1]$ or $x[t+1]$ to go outside its allowed range, then $r[t+1]$ is -1. The failure is said to occur at time $t+1$,

and the length of the trial is said to be t steps.

In addition to the reinforcement signal, the learning system also receives stimuli from the environment providing information about the current state of the cart-pole system. The stimulus representation is the same as that used by Michie and Chambers in their "Boxes" system (Michie and Chambers, 1968a,b) to solve this pole-balancing task. The 4-dimensional state space is divided into disjoint regions by quantizing the four state variables. Michie and Chambers distinguished 3 grades of cart position, 6 of pole angle, 3 of cart velocity, and 3 of pole angular velocity. The same number of quantizations are used here, but with different quantization thresholds than those used by Michie and Chambers:

$$x: \ \pm 0.8, \pm 2.4\,\mathrm{m},$$

$$\theta: \ 0, \pm 1, \ \pm 6, \ \pm 12\,\text{degrees},$$

$$\dot{x}: \ \pm 0.5, \ \pm\infty\,\mathrm{m/sec},$$

$$\dot{\theta}: \ \pm 50, \ \pm\infty\,\mathrm{deg/sec}.$$

This yields $3 \times 3 \times 6 \times 3 = 162$ regions corresponding to all combinations of the intervals. These values and units were chosen so as to produce what seemed like a physically realistic control problem, given the parameterization of the cart-pole simulation (Michie and Chambers did not publish the parameters of their cart-pole simulation).

At any time t the state of the cart-pole system is in exactly one of the 162 regions. Stimuli are considered to be 162-component vectors, all components of which were zero except for that corresponding to the region of the state space containing the current state of the cart-pole system. The Boxes learning system was designed specifically for this task and requires such an independent-associations approach. This approach is used here, even though the other algorithms do not require it, so that performance on this task can be compared with that attained

by the Boxes learning system. The Boxes learning system is not described here; the interested reader is referred to Michie and Chambers's papers (Michie and Chambers, 1968a,b).

The cart-pole system was simulated so as to match as closely as possible the behavior of a real physical system, including non-linearities and friction. The discrete-time equations and all parameter values used in the simulation are given in Table 8.

Some modifications were made to the learning algorithms to let them take advantage of the episodic nature of the task. At the beginning of each trial, all eligibility traces were set to zero, so that actions taken on the preceding trial were not assigned credit for the outcome of the new trial. For those algorithms maintaining a reinforcement-association vector $\vec{v}$, the traces used to update it were also set to zero after failures. Finally, $\hat{r}$ in the AHC algorithm was computed slightly differently at the beginning and end of each trial. At the beginning of each trial, the first $\hat{r}$ value was given by

$$\hat{r}[1] = r[1] + \gamma p^0[1],$$

i.e., the subtractive p term that would otherwise be due to the preceding trial is omitted. Similarly, in the last $\hat{r}$ value of a trial, the additive p term that would otherwise be due to the next trial was left off:

$$\hat{r}[t+1] = r[t+1] - p^t[t],$$

where the failure occurred at $t+1$. These modifications were also made to the SS ($\gamma = 1$) algorithm applied to this task.

Computational expense prevented an exhaustive search of the parameter space of the algorithms on this task. A long series of preliminary runs were made with various parameter values for Algorithms 4, 5, 8, 9, 12, 13, 14, and 15 of this chapter

Table 8. Details of Cart-Pole Simulation.

$$x[t+1] = x[t] + \tau\dot{x}[t] \qquad \dot{x}[t+1] = \dot{x}[t] + \tau\ddot{x}[t]$$

$$\ddot{x}[t] = \frac{F[t] + m_p l\left[\dot{\theta}^2[t]\sin\theta[t] - \ddot{\theta}[t]\cos\theta[t]\right] - \mu_c \mathrm{sgn}(\dot{x}[t])}{m_c + m_p}$$

$$\theta[t+1] = \theta[t] + \tau\dot{\theta}[t] \qquad \dot{\theta}[t+1] = \dot{\theta}[t] + \tau\ddot{\theta}[t]$$

$$\ddot{\theta}[t] = \frac{g\sin\theta[t] + \cos\theta[t]\left[\frac{-F[t] - m_p l\dot{\theta}^2[t]\sin\theta[t] + \mu_c \mathrm{sgn}(\dot{x}[t])}{m_c + m_p}\right] - \frac{\mu_p\dot{\theta}[t]}{m_p l}}{l\left[\frac{4}{3} - \frac{m_p\cos^2\theta[t]}{m_c + m_p}\right]}$$

where:

$$\mathrm{sgn}(x) = \begin{cases} +1, & \text{if } x \geq 0; \\ -1, & \text{if } x < 0. \end{cases}$$

$g = -9.8$ meters/second^2, acceleration due to gravity,

$m_c = 1.0$ kilograms, mass of cart,

$m_p = 0.1$ kilograms, mass of pole,

$l = 0.5$ meters, half pole length,

$\mu_c = 0.0005$, coefficient of friction of cart on track,

$\mu_p = 0.000002$, coefficient of friction of pole on cart,

$F[t] = \pm 10.0$ newtons, force applied to cart's center of mass at time t,

$\tau =$ time in seconds corresponding to one simulation time step.

and the Boxes algorithm. It was found that the Boxes algorithm performed best on this task with its parameters set near the values used by Michie and Chambers. It was found that the even-numbered algorithm of each pair of algorithms performed better than the odd-numbered algorithm, and that it did so at very large values for the learning-rate parameter α in comparison to σ_y. At such large α values, behavior is almost always nearly deterministic, so that $y[t] - \pi[t]$ is almost always zero, which may explain the poor performance of the odd-numbered algorithms. No parameter values were found enabling Algorithms 4, 5, 8, or 9 to perform anywhere near as well as the Boxes algorithm. However, it proved relatively easy to find parameter values at which Algorithm 12 performed much better than the Boxes algorithm.

Figures 47, 48, and 49 show the results of an experiment in which Algorithms 12, 14, and the Boxes algorithm were applied to this task. The plots of Figures 47 and 48 are averages of performance over the 10 runs that produced the individual graphs shown in Figure 49. Figure 47 is plotted on a logarithmic scale, Figure 48 on a linear scale. In both figures, a single point is plotted for each bin of 5 trials giving the number of time steps until failure averaged over those 5 trials. 8 of the runs with Algorithm 12, and 2 of the runs with the Boxes algorithm, were terminated after 500,000 time steps before all 100 trials took place (those whose plots terminate short of 100 trials in Figure 49). The simulation was terminated before failure on the last trials of these runs. To produce the averages for all 100 trials shown in Figures 47 and 48, special provision was made for the so-interrupted runs. If the duration of the trial underway when the run was interrupted was less than the duration of the immediately preceding (and therefore complete) trial, then fictitious remaining trials were assigned the duration of that preceding trial. Otherwise, fictitious remaining trials were assigned the duration of the last trial when it was interrupted. This was done to prevent any short interrupted trials from producing deceptively low averages.

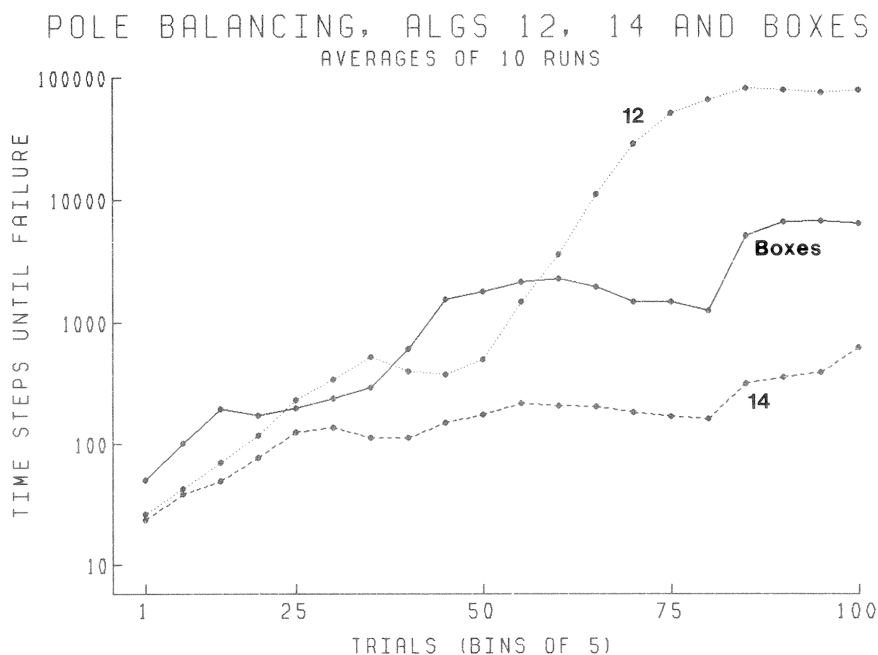


Figure 47. Average Performance of 3 Best Algorithms on Pole-Balancing Task, Logarithmic Scale. Each line shows the average performance over 10 runs of a particular algorithm over 100 trials; the label indicates the associated algorithm. Performance has been smoothed by averaging over bins of 5 trials.

The Boxes algorithm was run with the parameter settings published by Michie and Chambers (1968a). Algorithm 12 was run with a parameter setting which had shown good performance in the preliminary runs. These values were $\alpha = 1000$, $\gamma = .95$, $\beta = .5$, $\sigma_y = .01$, $\delta_w = .1$, and $\delta_v = .2$. Algorithm 14 was run with the same parameter settings as Algorithm 12.

Algorithm 12 achieved much longer runs than did the Boxes algorithm. Figure 49 shows that Algorithm 12 tended to solve the problem before it had experienced 100 failures, whereas the Boxes algorithm tended not to. The preliminary study showed that all other algorithms perform worse than Algorithm 14 did on this task.

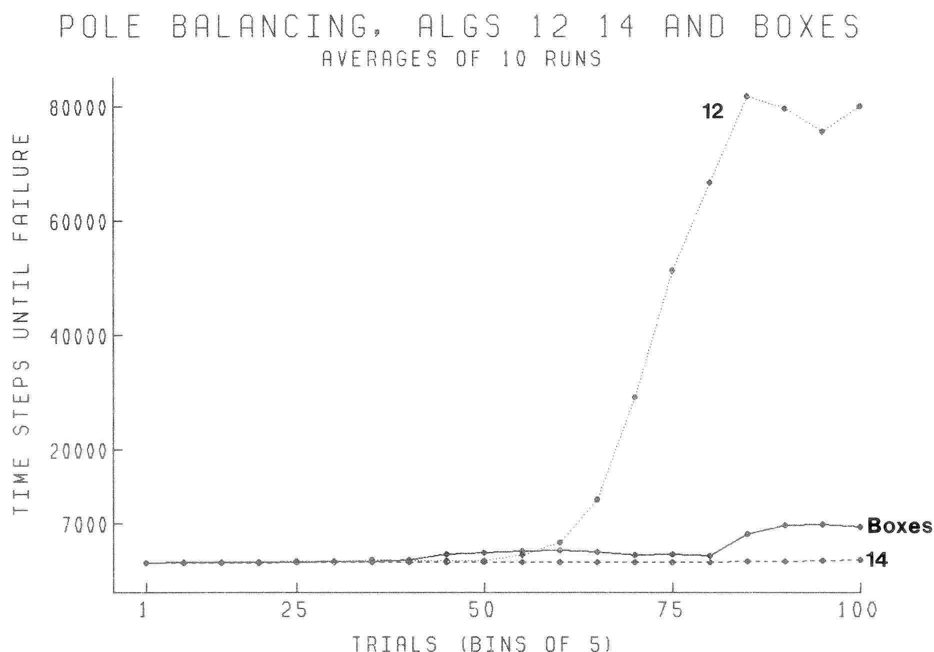


Figure 48. Average Performance of 3 Best Algorithms on Pole-Balancing Task, Linear Scale. Each line shows the average performance over 10 runs of a particular algorithm over 100 trials; the label indicates the associated algorithm. Performance has been smoothed by averaging over bins of 5 trials.

Summary

In experiments with simple abstract tasks and with a more complex and realistic pole-balancing task, it has been demonstrated that the AHC algorithm can improve performance. The capabilities of the AHC algorithm have been gauged by comparing its performance with that of the Boxes system (Michie and Chambers, 1968a,b), a learning algorithm designed specifically for the pole-balancing task and which does not involve secondary reinforcement. Although the Boxes algorithm performed well on this task, much better than the simplified version of Samuel's

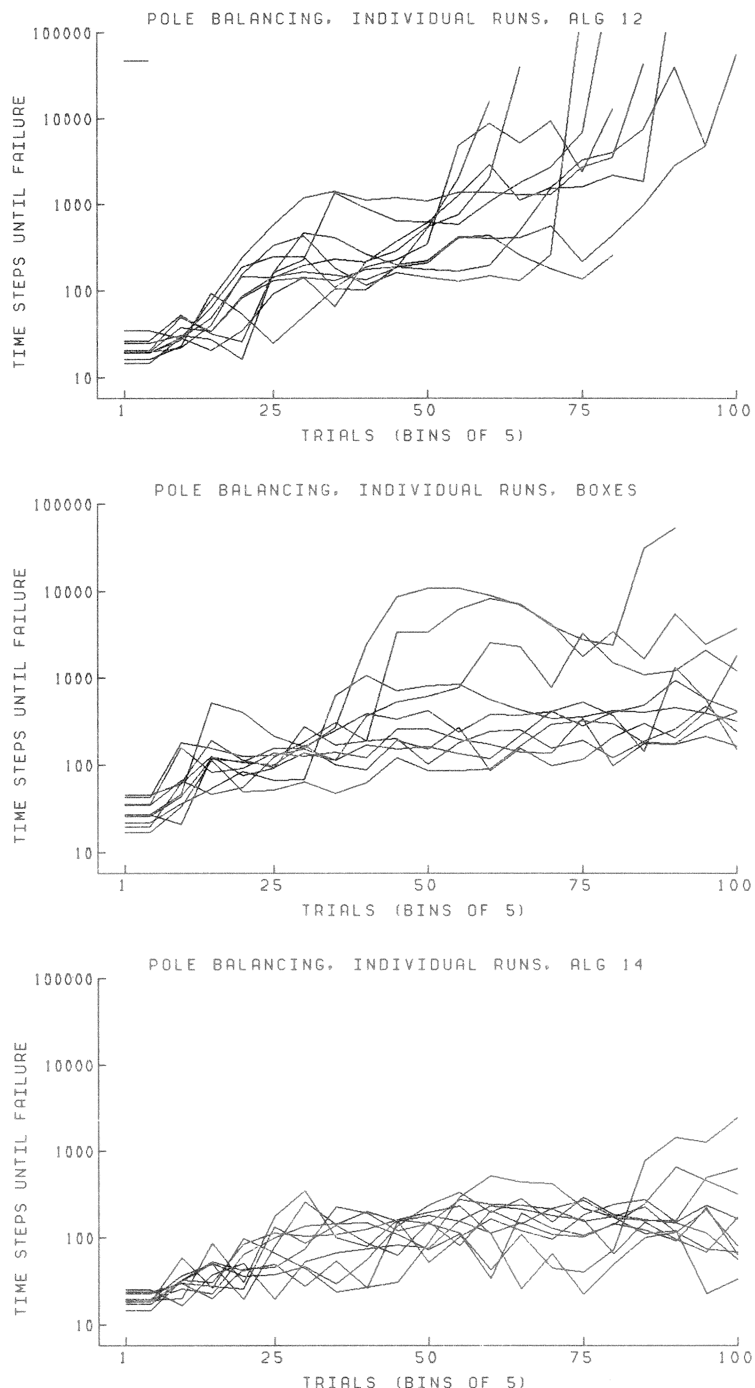


Figure 49. Individual Runs of 3 Best Algorithms on Pole-Balancing Task. From top to bottom, these graphs show the individual runs with Algorithm 12, the Boxes algorithm, and Algorithm 14. Each line shows the performance during one run of a particular algorithm. Performance has been smoothed by averaging over bins of 5 trials.

algorithm, the AHC algorithm performed much better still.

The AHC algorithm has also been compared with pure reinforcement-comparison algorithms, with algorithms lacking heuristic-reinforcement mechanisms, and with an algorithm due to Witten. Witten's algorithm is very closely related to the AHC algorithm, but does not compare reinforcement levels. The poor performance of Witten's algorithm highlights the need for reinforcement-comparison mechanisms, and the poor performance of the pure reinforcement-comparison mechanisms highlights the need to combine them with secondary reinforcement.

The experiments of this chapter have also revealed some major weaknesses of simple algorithms with eligibility factors based on $y[t] - \frac{1}{2}$. The use of the AHC algorithm seems to greatly ease these problems, at least in the tasks studied here. This observation may turn out to be critical given that the $y[t] - \frac{1}{2}$ form of this algorithm (as well as of the other algorithms) performed much better than the $y[t] - \pi[t]$ form on one of the tasks (the pole-balancing task).

CHAPTER VII

CLOSING

The problems of temporal credit assignment have been investigated within the domain of simple reinforcement-learning tasks. After systematic experimentation with a set of algorithms over a wide range of specific tasks, progress can be claimed in three areas 1) a number of temporal credit-assignment issues have been isolated and elucidated, including those of unbalanced reinforcement, misleading generalizations, delayed reinforcement, and secondary reinforcement, 2) it has been demonstrated that substantial performance improvements are possible through the use of sophisticated temporal credit-assignment algorithms, and 3) a new temporal credit-assignment algorithm, the AHC algorithm, has been presented and shown to perform better than previously considered algorithms on a variety of tasks.

Reinforcement Comparison

Reinforcement-comparison mechanisms have played a prominent role in this study. In Chapter II it was found that a learning algorithm's performance could be significantly improved through the use of mechanisms that remember past reinforcement levels and compare them with current reinforcement. The possibility of obtaining the same advantages in more complex tasks has been a major factor mo-

tivating the search for more sophisticated temporal credit-assignment algorithms throughout the rest of this dissertation.

The utility of reinforcement-comparison mechanisms on nonassociative reinforcement-learning tasks in which the action space is continuous is well recognized; *all* algorithms considered in the literature for these tasks are reinforcement-comparison algorithms. Most research with discrete-action tasks, on the other hand, has excluded reinforcement-comparison algorithms from consideration. In Chapter II, some simple but novel reinforcement-comparison algorithms for discrete-action nonassociative tasks are presented. The experiments of this chapter show that reinforcement-comparison mechanisms significantly improve learning rate on unbalanced-reinforcement tasks, i.e., tasks in which the distribution of positive and negative reinforcement values is not balanced. Overall, the new algorithms perform better than all others considered, including several from the learning automata theory literature.

To the best of my knowledge, reinforcement-comparison mechanisms have not previously been proposed for associative learning, except for the degenerate case in which no stimuli are similar. In Chapter III several novel reinforcement-comparison algorithms for associative reinforcement learning are presented. In a series of experiments involving similar stimuli and misleading generalizations, the new algorithms are shown to perform better overall than a variety of other algorithms. These experiments suggest that reinforcement-comparison mechanisms can ease the problems of misleading generalization as well as those of unbalanced reinforcement.

Delayed reinforcement creates such a difficult temporal credit-assignment problem by itself that further difficulties due to unbalanced reinforcement and misleading generalizations are sometimes overlooked. Accordingly, the advantages of reinforcement-comparison algorithms in delayed-reinforcement tasks are sometimes overlooked as well. Chapter IV's experiments with delayed-reinforcement

tasks show that reinforcement-comparison mechanisms can make significant performance improvements on delayed-reinforcement tasks.

Chapter VI concerns nonautonomous-stimuli tasks, i.e., tasks in which the learning system's behavior affects its subsequent non-reinforcing stimuli. Although reinforcement-comparison mechanisms substantially increase learning rate on the tasks considered in Chapter II–IV, on the more-general tasks of Chapter VI they actually hinder learning when used alone. On the tasks of this chapter, the algorithms with both reinforcement-comparison and secondary-reinforcement mechanisms perform best, yet the algorithms with *either mechanism alone* perform worse than those with neither. Although these two aspects of a temporal credit-assignment algorithm—reinforcement comparison and secondary reinforcement—can be teased apart and studied in isolation, their fortunes are intertwined. Apparently, the advantages that reinforcement-comparison mechanisms provide on autonomous-stimuli tasks can be obtained on more general tasks only by the concurrent use of secondary-reinforcement mechanisms.

Ideal Reinforcement Signal

The ideal reinforcement signal is the signal that provides evaluation of a learning system's behavior of the highest quality possible. In Chapter V, temporal credit-assignment algorithms are analyzed as attempts to approximate the ideal reinforcement signal. This theoretical construct assists the understanding of existing temporal credit-assignment algorithms and the creation of new algorithms. For example, from the concept of the ideal reinforcement signal it can be seen that a temporal credit-assignment algorithm should involve both comparison of evaluations and anticipation of delayed evaluations, i.e., both reinforcement-comparison and secondary-reinforcement mechanisms.

Chapter V presents several informal derivations of temporal credit-assignment algorithms as approximations to the ideal reinforcement signal. From one set of assumptions about the goal of learning, the AHC algorithm is derived, and from a slightly different set of assumptions, a simplified version of the temporal-credit assignment algorithm used in Samuel's checker-playing program (Samuel, 1959, 1967) is derived.

The AHC Algorithm and Samuel's Checker-Playing Program

One reason for the success of Samuel's celebrated checker-playing program (Samuel, 1959, 1967) is its "learning-by-generalization" mechanism for temporal credit assignment. Despite wide recognition of the merits of the approach, there has been little follow-up since Samuel's original work. If Samuel's algorithm is simplified in several ways, including the removal of all features specialized for the game of checkers, for tasks in which off-line search is performed, and for efficient implementation on a small computer, the resulting algorithm differs from the AHC algorithm only by the absence of one parameter, called the discount-rate parameter. In this sense the AHC algorithm can be viewed as a generalization of Samuel's algorithm.

Theoretical analysis in terms of the ideal reinforcement signal suggests that although the discount-rate parameter may not be needed for the special class of time-blind tasks (which includes checker playing), it may be necessary for others. Experiments with simple abstract tasks and with a physically-realistic pole-balancing task support this conclusion by showing improved performance attributable to the use of the discount-rate parameter. On the pole-balancing task the AHC algorithm performs markedly better than both Samuel's algorithm and an algorithm specifically designed for this task by Michie and Chambers (1968a,b) which does not use

secondary reinforcement. Samuel's algorithm, on the other hand, performs worse than Michie and Chambers's algorithm.

Limitations and Future Work

The investigation of temporal credit assignment begun here is far from complete. Although many tasks and algorithms have been considered, they do not begin to exhaust the space of possibilities. Chapters IV and VI in particular investigate only a small subset of possible delayed-reinforcement and nonautonomous-stimuli tasks. Some of the possibilities not treated are mentioned at the end of each chapter.

As a study of credit assignment in reinforcement learning, this dissertation is limited by the lack of attention it gives to structural credit assignment. Great care has been taken here to separate the temporal and structural credit-assignment problems, and to concentrate as fully as possible on the former. Dividing the credit-assignment problem in this way has permitted its temporal aspects to be investigated more thoroughly than would otherwise have been possible, but this division has also resulted in virtually nothing being learned about structural aspects of the credit-assignment problem. All temporal credit-assignment algorithms considered here can be viewed as attempts to approximate the ideal reinforcement signal, yet even the ideal reinforcement signal is useless for structural credit assignment.

Possible future work concerning the structural credit-assignment problem includes comparatively simple extensions, such as going beyond linear maps to consider nonlinear maps of fixed structure, or going beyond binary-action tasks to consider n-action tasks or tasks with continuous action spaces. More novel would be algorithms that determine which stimuli and actions are most likely to be respon-

sible for outcomes, and which then assign these greater structural credit. Finally, and potentially most importantly, credit-assignment methods need to be found that can direct the *structural* modification of representations and decision processes.

The structural credit-assignment problem should be a fruitful area for future work. Dividing the credit-assignment problem into temporal and structural subproblems has aided this investigation of the temporal subproblem, and may aid investigations of the structural subproblem in the future. To the extent that the two subproblems are truly separable, it should be possible to combine, without modification, separately developed solutions to each.

Applications

The temporal credit-assignment algorithms discussed in this dissertation have a wide range of potential applications. As long as performance is not limited entirely by structural credit-assignment factors, these algorithms may be usefully applied to almost any task involving evaluative feedback. If a task involves delayed evaluations that can be anticipated, misleading generalizations, or different levels of attainable evaluations at different times (unbalanced reinforcement), then it involves temporal credit-assignment problems that can be eased by these algorithms. Such applications arise frequently but do not exclusively occur in reinforcement-learning tasks.

Reinforcement learning has many applications in control problems, e.g., setting fuel mixtures or other controls for engines, turbines, manufacturing processes or communication networks. Reinforcement learning is applicable, while traditional control-theory approaches are not, wherever performance can be measured but information indicating correct behavior cannot. This is often the case when a good model of the system to be controlled is not available or is more expensive to acquire

than the cost of the time spent in learning. Mendel and McLaren (1970) briefly discuss several applications for reinforcement learning control systems along these lines.

Incomplete knowledge of the sort that makes reinforcement learning applicable is common in the control problems that arise in robotics. Although detailed models of the robots themselves can be constructed, these generally rely on a restricted class of motions and control strategies, e.g., nonballistic motion. Even so, much of the robot's environment, such as the position of objects to be manipulated, is rarely known with complete reliability. Extensive contingency planning, or trial-and-error with human supervision and intervention, is generally required before a robot can be relied upon to successfully perform even simple tasks. In many cases the process might be considerably expedited if goals could be specified in addition to motions, leaving it it to a learning system to handle exceptional cases.

Associative reinforcement learning could also be used to aid heuristic search. The efficiency of heuristic search is highly dependent on the heuristics used to select promising alternatives for further search. An associative reinforcement-learning system could be used to make these selections based on features of the source node. For example, suppose the search tree of a board game is to be searched. A learning algorithm could suggest moves to explore from each node, based on their characteristics and on the board position, and have its suggestions evaluated according to search success along the selected paths. Heuristics learned in this way might greatly reduce search time. And a temporal credit-assignment mechanism, such as the AHC algorithm, would almost certainly greatly aid such a "learning heuristic search."

Temporal credit-assignment algorithms also have potential applications that do not involve reinforcement learning. For example, the AHC algorithm could be used to aid identification of faulty components in complex knowledge-based systems.

The AHC algorithm could help identify those steps in a sequences of processing steps in which faults occur, thus reducing the number of possible culprits.

Finally, and more speculatively, the AHC algorithm could be used to anticipate changes in economic indicators such as the Dow Jones average. The AHC algorithm's ability to improve immediacy, to bring forthcoming changes into the present, is just the sort of capability required. In addition, the AHC algorithm's linear-mapping approach allows it to combine many factors (albeit linearly) to make an overall prediction, which would clearly be required for this application.

Reinforcement Learning

It is sometimes thought that reinforcement learning was thoroughly investigated in the 1950's and 1960's, and that only small incremental improvements in performance could be attained by further work in this area. This dissertation's experiments suggest the opposite. Large differences in the capabilities of reinforcement-learning algorithms have been observed repeatedly in these experiments. Some algorithms learn much more slowly than others, some do not work at all outside of a limited range of tasks, and some fail spectacularly on simple tasks with special properties. Associative reinforcement learning involves subtle difficulties that have not been elucidated by earlier research.

REFERENCES

Anderson, C. W. Feature generation and selection by a layered network of reinforcement learning elements: Some initial experiments. COINS Technical Report 82-12, University of Massachusetts, Amherst, 1982.

Anderson, J.A., Silverstein, J.W., Ritz, S.A. & Jones, R.S. Distinctive features, categorical perception, and probability learning: Some applications of a neural model. *Psychological Review*, 1977, *85*, 413–451.

Atkinson, R.C., Bower, G.H. & Crothers, E.J. An Introduction to Mathematical Learning Theory. New York: Wiley, 1965.

Atkinson, R.C. & Estes, W.K. Stimulus sampling theory. In R.D. Luce, R.R. Bush & E. Galanter, Eds. *Handbook of mathematical psychology, Vol. II.* New York: Wiley, 1963.

Barr, A,. & Feigenbaum, E.A., Eds. *The Handbook of Artificial Intelligence.* Los Altos, CA: Kaufman, Inc, 1981.

Barto, A.G., Ed. Simulation experiments with goal-seeking elements, in press.

Barto, A. R., Anderson, C. W. & Sutton, R. S. Synthesis of nonlinear control surfaces by a layered associative search network. *Biological Cybernetics*, 1982, *43* 175–185.

Barto, A. G. & Sutton, R. S. Goal seeking components for adaptive intelligence: An initial assessment. *Air Force Wright Aeronautical Laboratories/Avionics Laboratory Technical Report AFWAL-TR-81-1070*, Wright-Patterson AFB, Ohio, 1981.

Barto, A. G. & Sutton, R. S. Simulation of anticipatory responses in classical

conditioning by a neuron-like adaptive element. *Behavioral Brain Research*, 1982, *4* 221–235.

Barto, A. G. & Sutton. R. S. Landmark learning: An illustration of associative search. *Bioliological Cybernetics*, 1981, *42*, 1–8.

Barto, A. G., Sutton, R. S. & Brouwer, P. S. Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, 1981, *40*, 201–211.

Barto, A.G., Sutton R.S. & Anderson, C.W. Neuronlike elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, in press.

Booker, L.B. Intelligent behavior as an adaptation to the task environment. Univ. of Michigan Ph.D. dissertation, 1982.

Bradt, R. N, Johnson, S. M. & Karlin, S. On sequential designs for maximizing the sum of n observations. *Ann. Math. Stat.*, 1956, *27*, 1060–1074.

Buchanan, B.G., Smith, D.H., White, W.C., Gritter, R.J., Feigenbaum, E.A., Lederberg, J. & Djerassi, C. Automatic rule formation in mass spectroscopy by means of the Meta-DENDRAL program. *J. Am. Chem. Soc.*, 1976, *98*.

Buchanan, B.G., Mitchell, T.M. & Smith, R.G. Models of learning systems. In *Encyclopedia of Computer Science and Technology*, edited by J. Belzer. New York: Marcel Dekker, Inc., 1978.

Bush, R. R. & Estes, W. K., Eds. *Studies in Mathematical Learning Theory.* Stanford: Stanford University Press, 1959.

Bush, R. R. & Mosteller, F. *Stochastic Models for Learning.* New York: Wiley, 1955.

Carbonell, J.G. Learning by analogy: Formulating and generalizing plans from past experience. In *Machine Learning*, edited by R.S. Michalski, J.G. Carbonell & T.M. Mitchell. Palo Alto, CA: Tioga Pub. Co., 1982.

Carterette, T. S. An application of stimulus sampling theory to summated generalization. *Journal of Experimental Psychology*, 1961, *62*, 448–455.

Cover, T. M. & Hellman, M. E. The two-armed bandit problem with time-invariant finite memory. *IEEE Transactions on Information Theory*, 1970, *16*, 185–195.

Derman, C. *Finite State Markovian Decision Processes.* New York: Academic Press, 1970.

Dietterich, T.G., London, B., Clarkson, K. & Dromey, G. Learning and inductive inference. Chapter XIV of the *Handbook of Artificial Intelligence, Vol III*, edited by Paul R. Cohen & Edward A. Feigenbaum. Los Altos, CA: William Kaufman, Inc., 1982. (Also Stanford Computer Science Dept. Tech. Rep. STAN-CS-82-913.)

Dietterich, T.G. & Michalski, R.S. Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods. *Artificial Intelligence*, 1981, *16*, 257–294.

Duda, R.O. & Hart, P.E. *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

Estes, W.K. Toward a statistical theory of learning. *Psychololgical Review*, 1950, *57*, 94–107.

Estes, W.K. The statistical approach to learning theory. In S. Koch, Ed., *Psychology: A Study of a Science, Vol. 2.* New York: McGraw-Hill, 1959a.

Estes, W.K. Component and pattern models with Markovian interpretations. In R. R. Bush & W. K. Estes, Eds., *Studies in Mathematical Learning Theory.* Stan-

ford: Stanford University Press, 1959b.

Farley, B. G. & Clark, W. A. Simulation of self-organizing systems by digital computer. *I.R.E. Transactions on Inf. Theory*, 1954, *4*, 76–84.

Feldman, J. A. A connectionist model of visual memory. In Hinton, G. & Anderson, J., Eds., *Parallel Models of Associative Memory*. Hillsdale, NJ: Erlbaum, 1981.

Feldman, J.A. Dynamic connections in neural networks. *Biological Cybernetics*, 1982, *46*, 27–39.

Fogel, L.J., Owens, A.J. & Walsh, M.J. *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.

Friedberg, R.M. A learning machine, part I. *IBM Journal of Research and Development*, January, 1958, *2*, 2–13.

Friedberg, R.M., Dunham, B. & North, J.H. A learning machine, part II. *IBM Journal of Research and Development*, June, 1959, *3*, 282–287.

Friedman, M. P., Trabasso, T. & Mosberg, L. Tests of a mixed model for paired-associates learning with overlapping stimuli. *Journal of Mathematical Psychology*, 1967, *4*, 316–334.

Hampson, S. & Kibler, D. A boolean complete neural model of adaptive behavior. UC Irvine, Dept. of Information and Computer Science Technical Report #190, Nov., 1982.

Holland, J.H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. of Michigan Press, 1975.

Holland, J.H. Adaptive knowledge acquisition. In preparation.

Howland, B., Minsky, M.L. & Selfridge, O.G. Hill-climbing: Some remarks on

multiple simultaneous optimization. Group Report 54-15, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA, 1960.

Jarvis, R.A. Adaptive global search in a time-variant environment using a probabalistic automaton with pattern recognition supervision. *IEEE Transactions on Systems Science and Cybernetics*, 1970, *Vol. SSC-6*, No. 3.

Jarvis, R.A. Optimization strategies in adaptive control: A selective survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 1975 *Vol. SMC-5*, No. 1.

A. H. Klopf. Brain function and adaptive systems– A heterostatic theory. *Air Force Cambridge Research Laboratories Research Report*, AFCRL-72-0164, Bedford, MA., 1972 (A summary appears in *Proceedings International Conference on Systems, Man, Cybernetics*). *IEEE Systems, Man, and Cybernetics Society*, 1974, Dallas, Texas.

Klopf, A. H. *The hedonistic neuron: A theory of memory, learning, and intelligence.* Washington, D.C.: Hemisphere, 1982.

La Berge, D. L. Generalization gradients in a discrimination situation. *Journal of Experimental Psychology*, 1961, *62*, 88–94.

Lakshmivarahan, S. *Learning Algorithms and Applications.* Springer-Verlag, 1981.

Langley, P. & Simon, H.A. The central role of learning in cognition. In: *Cognitive Skills and their Acquisition*, edited by J.R. Anderson. Hillsdale, NJ: Lawrence Erlbaum Associates, 1981.

Lenat, D.B. AM: An artificial intelligence approach to discovery in mathematics as heuristic search. Rep. No. STAN-CS-76-570, Computer Science Dept., Stanford University, 1976. (Doctoral dissertation. Reprinted in *Knowledge-based Systems in Artificial Intelligence*, edited by R. Davis & D.B. Lenat. New York: McGraw-Hill,

1980.)

Lenat, D. B., Hayes-Roth, F., Klahr, P. Cognitive economy. Stanford Heuristic Programming Project HPP-79-15 (working paper), 1979.

Lovejoy, E. *Attention in Discrimination Learning.* San Francisco: Holden-Day, 1968.

Luce, R. D., Bush, R. R & Galanter, E., Eds. *Handbook of Mathematical Psychology*, Vols. I and II. New York: Wiley, 1963.

Luce, R. D., Bush, R. R. & Galanter, E., Eds. *Handbook of Mathematical Psychology*, Vol. III. New York: Wiley, 1965.

Marrs, P. & Poppelbaum, E. J. *Stochastic and deterministic averaging processors.* London: The Institute of Electrical Engineers, 1981.

Mason, L.G. An optimal S-model learning algorithm for S-model learning environments. *IEEE Trans. on Automatic Control*, 1973, 493–496.

McMurtry, G.J. Adaptive optimization procedures. In: *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, edited by Mendel, J.M. & Fu, K.S., pp. 243–286. New York: Academic Press, 1970.

Mendel, J.M. A survey of learning control systems. *ISA Transactions*, 1966, *5*, 297–303.

Mendel, J. M. & McLaren, R. W. Reinforcement learning control and pattern recognition systems. In: *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, edited by J.M. Mendel & K.S. Fu, pp. 287–318. New York: Academic Press, 1970.

Michie, D. & Chambers, R. A. BOXES: An experiment in adaptive control. In

Dale, E. & Michie, D., Eds., *Machine Intelligence 2.* Edinburgh: Oliver and Boyd, 1968a, 137–152.

Michie, D. & Chambers, R. A. 'Boxes' as a model of pattern-formation. In Waddington, C.H., Ed., *Towards a Theoretical Biology; 1, Prolegomena.* Edinburgh: Edinburgh University Press, 1968b, 206–215.

Mine, H. & Osaki, S. *Markovian Decision Processes.* New York: American Elsevier Publishing Company, Inc., 1970.

Minsky, M. L. Theory of neural-analog reinforcement systems and its application to the brain-model problem. Princeton Univ. Ph.D. Dissertation, 1954.

Minsky, M. L. Steps toward artificial intelligence. *Proceedings IRE*, 1961, *49*, 8–30. (Reprinted in Feigenbaum, E. A. & Feldman, J., Eds., *Computers and Thought.* New York: McGraw-Hill, 1963, 406–450.)

Minsky, M.L. K-lines: A theory of memory. *Cognitive Science*, 1980, *4*, No. 2, 117–133.

Minsky, M. L. & Papert, S. *Perceptrons: An introduction to computational geometry.* Cambridge, MA: MIT Press, 1969.

Minsky, M.L. & Selfridge, O.G. Learning in random nets. In: *Information Theory: Fourth London Symposium*, edited by C. Cherry. London: Butterworths, 1961.

Mitchell, T.M. Version spaces: An approach to concept learning. Doctoral dissertation, Stanford University, Stanford, CA, 1978.

Mitchell, T.M., Utgoff, P.E., Nudel, B. & Banerji, R.B. Learning problem-solving heuristics through practice. *IJCAI*, 1981, *7*, 127–134. (see also Mitchell et.al., 1973)

Mitchell, T.M., Utgoff, P.E. & Banerji, R.B. Learning problem-solving heuristics by experimentation. In: *Machine Learning*, edited by R.S. Michalski, T.M. Mitchell & J. Carbonell. Palo Alto, CA: Tioga, 1983.

Narendra, K.S. & Thathachar, M.A.L. Learning automata—a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 1974, *4*, 323–334.

Newell, A. The chess machine. In *Proceedings of the 1955 Western Joint Computer Conference*, 1955, session on Learning Machines, pp. 101–108, W.H. Ware, chairman.

Nilsson, N.J. *Learning Machines.* New York: McGraw-Hill, 1965.

Poggio, T. 1975. On optimal nonlinear associative recall. *Biol. Cybernetics*, 1975, *19*, 201–209.

Raibert, M. H. A model for sensorimotor control and learning. *Biological Cybernetics*, 1978, *29*, 29–36.

Reilly, D. L., Cooper, L. N. & Elbaum, C. A neural model for category learning. *Biological Cybernetics*, 1982, *45*, 35–41.

Restle, F. A theory of discrimination learning. *Psychological Review*, 1955, *62*, 11–19.

Rissland, E.L. & Soloway, E.M. Constrained example generation: A testbed for studying issues in learning. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981, p. 162.

Rosenblatt, F. The perceptron: A perceiving and recognizing automaton. Rep. No. 85-460-1, Project PARA, Cornell Aeronautical Laboratory, Buffalo, NY, 1957.

Rosenblatt, F. *Principles of Neurodynamics.* New York: Spartan Books, 1962.

Samuel, A.L. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 1959, *3*, 210–229. (Reprinted in *Computers and Thought*, edited by E. A. Feigenbaum & J. Feldman, pp. 71–105. New York: McGraw-Hill, 1963.)

Samuel, A.L. Some studies in machine learning using the game of checkers. II–Recent progress. *IBM Journal of Research and Development*, 1967, *11*, 601–617.

Schank, R.C. *Dynamic Memory.* 1983

Selfridge, O.G. Pattern recognition and learning. yIn: *Proceedings of the 3d London Symposium on Information Theory*, edited by C. Cherry, p. 345. London: Butterworth; also New York: Academic, 1956.

Selfridge, O. G. Pandemonium: A paradigm for learning. *Proceedings of the Symposium on the Mechanisation of Thought Processes.* Teddington, England: National Physical Laboratory, H.M. Stationary Office, London, 2 vols., 1959.

Smith, S.F. A learning system based on genetic adaptive algorithms. Doctoral dissertation, University of Pittsburg, Pittsburg, Penn., 1980

Soloway, E. Learning = interpretation + generalization: A case study in knowledge-directed learning. Rep. No. COINS-TR-78-13, Computer and Information Sciences Dept., University of Massachusetts, Amherst, 1978. (Doctoral dissertation.)

Spinelli, D.N. OCCAM: A computer model for a content addressable memory in the central nervous system. In: *The Biology of Memory*, Pribram, K., Broadbent, D., eds. New York: Academic Press, 1970.

Sussman, G.J. A computational model of skill aquisition. AI Tech. Rep. 297, AI Laboratory, Massachusetts Institute of Technology, 1973. (Doctoral dissertation,

see also Sussman, 1975)

Sussman, G.J. *A Computer Model of Skill Aquisition.* New York: American Elsevier, 1975.

Sutherland, N.S. & Mackintosh, N.J. *Mechanisms of animal discrimination learning.* New York: Academic Press, 1971.

Sutton, R.S. A unified theory of expectation in classical and instrumental conditioning. Stanford undergraduate thesis, 1978.

Sutton, R.S. & Barto, A.G. Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review,* 1981, *88*, 135–171.

Tsetlin, M.L. On the behavior of finite automata in random media. *Automat. Telemekh.*, Oct., 1961, *22*, 1345–1354.

Tsetlin, M.L. *Automaton Theory and Modelling of Biological Systems.* New York: Academic Press, 1973.

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

Widrow, B. Generalization and information storage in networks of adaline "neurons". In Yovits, M., Jacobi, G. & Goldstein, G., Eds., *Self-organizing systems.* Spartan Books, 1962.

Widrow B. & Hoff, M. E. Adaptive switching circuits. *1960 WESCON Convention Record Part IV*, 1960, 96–104.

Witten, I.H. An adaptive optimal controller for discrete-time markov environments. *Information and Control,* 1977, *34,* 286–295.

Winston, P.H. Learning structural descriptions from examples. Rep. No. 231,

AI Laboratory, Massachusetts Institute of Technology, 1970. (Reprinted in *The Psychology of Computer Vision*, edited by P.H. Winston, pp. 157–209. New York: McGraw-Hill, 1975.)

Zeaman, D. & House, B.J. The role of attention in retardate discrimination learning. In Ellis, N.R., Ed., *Handbook of Mental Deficiency*. New York: McGraw-Hill, 1963.