CHAPTER 13

ARCHITECTURES FOR STRATEGY LEARNING*

Pankaj Mehra and Benjamin W. Wah

1 INTRODUCTION

Recent years have witnessed a sharp rise in the quantity and quality of knowledge that can be captured in automated systems. Artificial intelligence (AI) research in the 1980s has focused on the issue of automatic knowledge acquisition or *learning*. Learning tasks are perhaps the most complex problems that AI researchers attempt to solve. Concomitant developments in the area of artificial neural systems (ANSs) have also prompted research in machine learning—mainly by providing for learning systems an inherently parallel model of implementation, one in which generalization and constraint satisfaction are spontaneous. This chapter is an attempt to analyze and illustrate how various models (old and new) can be applied to complex learning tasks in strategic problem solving.

In his famous essay, *The Architecture of Complexity* [207], Simon outlined the nature of complexity in problem solving by comparing it to a search through a maze. Subsequently, Simon and Newell formalized the search space model of problem solving. From the very beginning of this work, it was clear that the complexity of a problem (and the associated search) was due mainly to the large number of alternatives at each step. Simon pointed out that

*This research was supported by the National Science Foundation under Grant MIP-88-10598, and by National Aeronautics and Space Administration under Grant NCC 2-481.

the trial and error [in exploring the alternatives] is not completely random or blind; it is in fact rather highly selective. . .The selectivity derives from various rules of thumb, or heuristics, that suggest which paths should be tried first and which leads are promising.

This treatment of problem solving led to work in the theory of heuristic search [166, 167], which resulted in very general models of problem solving on one hand and efficient, general algorithms for search on the other [252]. Research suggested that search algorithms draw their power from the expressiveness and efficiency of problem representation, as well as from search control knowledge encoded in heuristic rules. Work on expert systems later confirmed this assessment [85], and it was realized that overcoming the *knowledge acquisition bottleneck* was the key to designing powerful programs. With this realization, the focus in problem solving shifted from powerful search algorithms to learning of heuristic knowledge for problem solving.*

Various algorithms for learning have been proposed and implemented during the last decade and several general paradigms of learning have arisen. However, most of the algorithms were demonstrated on small, well-defined domains, such as game-playing [120, 144, 240], the blocks world [142], symbolic integration [147], and high school arithmetic [51, 206]. Furthermore, the complexity of the other applications was due more to the structure of their environments than to the dynamic variability of their parameters [148]. Our recent attempts at solving certain complex problems in resource allocation have been frustrated by the inadequate learning model assumed by existing algorithms. In order to correct this deficiency in a systematic way, we have

1. studied the origins of complexity in strategy learning, using various aspects of complexity to identify difficult problem classes,
2. analyzed the applicability of several well-known learning algorithms to various problem classes, and
3. proposed a model of learning systems that applies to a class of complex problems.

The rest of this section introduces the fundamental concepts of strategy learning systems.

1.1 Problem Solving Strategy

A problem solver has a variety of knowledge about its domain. The procedural component of this knowledge is available as primitive pieces of procedural code called *operators*. Each operator is defined by the way it transforms a problem situation (or *state*) into another. A strategy for solving a problem is

* The complementary question of automatic acquisition of problem representation and its relationship to the strategy acquisition process is beyond the scope of this chapter (See Section 1.7).

a body of abstract procedural knowledge stated in terms of operators. The problem itself is a generic description of several *problem instances*. It is stated in terms of parameters, constraints, and objectives. Each instance is defined by its initial situation, that is, by the initial assignment of values to parameters. A *solution* to a problem instance is typically stated as a partial order on a set of operators. When applied to the initial description, a solution generates another description that satisfies the constraints and meets the objective. A *problem-solving strategy*, in this respect, is a systematic method for generating solutions to the instances of a problem.

1.2 Strategic Knowledge

The *object level* search space of a strategy learning system is defined by the knowledge of the problem and its instances. The heuristics and strategies form the first level of *metaknowledge*. Traditionally, metaknowledge has been made available to programs directly [27] and accounts for much of their problem-solving capability. However, in some domains metaknowledge is essentially empirical and varies from one context to another; it is thus necessary to acquire this component automatically. The acquisition of metaknowledge is called *strategy learning*. Construction of strategies is accomplished using a process for learning strategic metaknowledge; the knowledge controlling the strategy-learning process can be called the *meta-metaknowledge* of the problem (Figure 13.1).

1.3 Learning

The study of learning is the study of adaptive systems, and for the last decade the modern view of learning systems has followed developments in cognitive science, artificial intelligence, and adaptive control systems theory. A learning

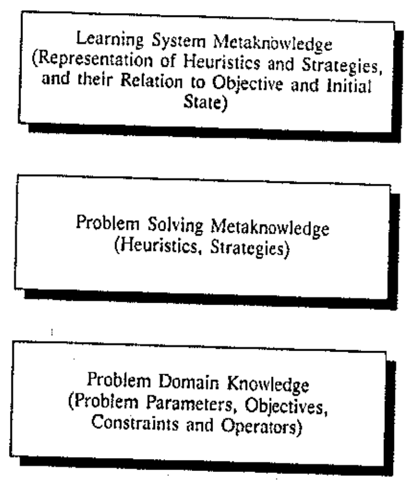

Figure 13.1 Hierarchy of knowledge in strategy-learning systems.

system adapts over time by making changes that improve its performance. Under Feigenbaum's knowledge hypothesis (Knowledge is power) [61, 128], learning has become synonymous with knowledge acquisition. For insightful discussions of the fundamental issues in machine learning, the reader is referred to articles by Simon [213], Minsky [140], Schank *et al.* [204], Langley [118], and Berliner [27].

1.4 Nomenclature of Strategy-Learning Paradigms

A *strategy-learning paradigm* is a way to represent and acquire metaheuristic knowledge for problem solving. Different learning paradigms make different assumptions about the learning process; these paradigms affect the architecture of the problem solvers, which, in turn, influences the performance of the overall system. Following Dietterich, we shall attempt to analyze strategy-learning systems at the knowledge level [54, 155]. In what follows, we view strategy learning as a search process; just as the first level of metaknowledge is useful in searching for solutions at the object level, so the second level of metaknowledge is useful in searching for heuristics and strategies at the first level. Second-level search is more difficult, however, because its objectives are determined only partially by the objectives of search at the object level. The constraints at the second level include the time taken by a strategy when it is applied to a problem instance, the quality of the object-level solutions produced by the strategy, and constraints on the applicability of strategies. The operators at this level are called *strategy modifiers* or *strategy constructors*. The metaheuristic knowledge involved in the process is called the *credit assignment policy*. The strategy modifiers constitute a body of knowledge known classically as the *learning element*; the metaheuristic knowledge is implemented in a component known classically as the *critic* (see next section; also see Chapter 14, vol. 3, [21]).

It is important that the critic be sensitive both to the objectives and constraints of the object-level problem and to the constraints of the meta-level problem. If the critic is implemented procedurally, the strategy-learning process is called *analytical*. If the critic's sensitivity to the domain-level objective is acquired by observing the effect of a domain-level strategy on a reactive environment, the process is called *empirical*. Finally, if the critic is implemented so that it has background knowledge of the relationship between the objectives at the object level and the explicitly represented solutions found by strategies at the meta-level, the process is said to be *knowledge-based*.

1.5 Architectures for Strategy-Learning Systems

The components of the problem-solving and learning system and their overall arrangement constitute the architecture of a strategy-learning system. Fundamental issues in the design of such architectures have been addressed by Dietterich [52], Minsky [139], and Mitchell [144, 215]. Our research has

caused us to reassess the architecture proposed by Dietterich; other architectural issues arise from recent developments in ANSs [1].

Dietterich's *classical model* has a learning element, a performance element, and a critic (see Figure 13.2). The performance element consists of the problem solver and associated bodies of knowledge, which include the basic search control rules, such as heuristics for selecting, ordering, and pruning of alternatives. At each decision point, the heuristic knowledge is invoked to determine what operator to apply to find the best path to the goal. Because the set of heuristics may be incomplete or inaccurate, the selected action may not be the best option; it is the job of the critic to evaluate either the action or, more commonly, the environment that results from its application. The evaluation is passed to the learning element, which edits the heuristic knowledge so that subsequent selections may yield better evaluations. Consequently, the learning element must know how the performance element works, including how it represents knowledge and what inference schemes it uses. This knowledge can be specified procedurally either in the form of a learning algorithm such as that used by LEX [146], or declaratively in the form of editing rules such as those in LS-1 [217], SAGE.2 [119], EURISKO [126], ODYSSEUS [246, 247], and Cupr [106]. Learning systems differ in the amounts of explicit and implicit knowledge that go into the design and interaction of these components.

A popular model of strategy learning proposed by Langley [120] is an instance of the Dietterich model. In it, the performance element is called the *behavior generation component*, the learning element is called the *behavior modification component*, and the critic is called the *behavior assessment component*. We shall refer to systems based on this model as belonging to the Dietterich model.

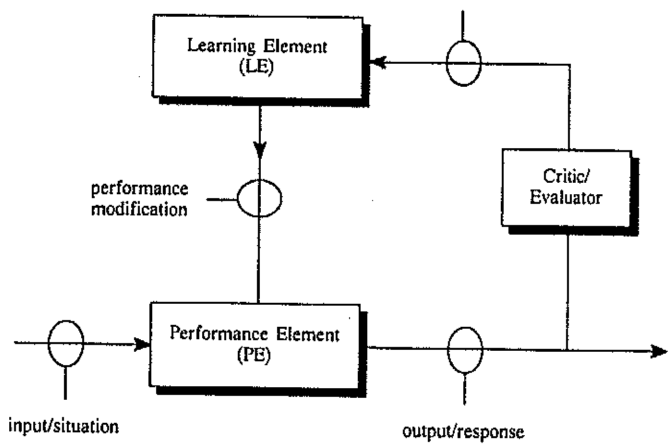

Figure 13.2 Dietterich's model of learning systems.

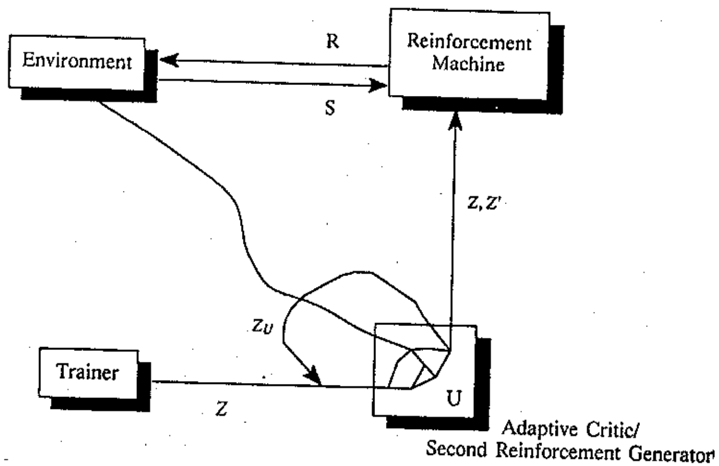

Figure 13.3 Minsky's reinforcement learning model.

An older and perhaps less restricted model of learning systems is that of Minsky [140] (see Figure 13.3).

Z is the external reinforcement; $Z = \pm 1$ for reward/extinction. The function $c_n = (2p_n - 1)$ is monotonic with the probability p_n of producing response R to stimulus S. The reinforcement machine tries to have a high value of p_n if Z_n is expected to be high, and vice versa. This can be achieved by adjusting c after every reinforced trial so that the correlation between c and Z approaches unity. The reinforcement machine adjusts c by maintaining a moving average of past reinforcements; for example,

$$c_{n+1} = (1 - \Theta) \sum_{i=0}^{n} \Theta^{n-i} Z_i, \text{ where } \Theta \to 0$$

which is the *exponentially decreasing average*, or

$$c_{n+1} = \left(1 - \frac{1}{N}\right) c_n + \frac{1}{N} Z_n, \text{ where } N \text{ varies with time}$$

which is the *uniform weighted average*. The term $\frac{1}{N}$ can be viewed as the *sensitivity* of the average to new data items.

z' is the expected reinforcement such that the probability $P(z = z'|E)$ is high, where E is the current state of the environment. Sometimes z may not follow R, in which case z' may be used. When z is available, the secondary reinforcement machine may be so adaptive that the difference between z and z' approaches zero. z_U is a positive feedback connection in the secondary reinforcement machine. It could be used to predict more than one step ahead, but positive feedback may cause instability.

The Dietterich model originated in knowledge-intensive AI systems, whereas the Minsky model originated in knowledge-lean domains. The Minsky model is distinctive because its critic is adaptive and requires only a small amount of implicit knowledge about its environment. This feature may sometimes be a drawback: if a partial internal model is available, the model has no way to use this knowledge. Samuel's Checker Player is a classic example of the Minsky model [202, 203], and other examples of the Minsky model include the REINFORCE model of Williams [248], the drive-reinforcement model of Klopf [104], and the reinforcement learning model of Barto et al. [22, 23, 226]. Even though this model of learning was proposed by Minsky, he was not convinced that such a machine could be at the heart of an intelligent system, except in the case of truly statistical environments. However, with recent advances in the ability to represent search behavior in such systems, the model has become much more applicable. We maintain that the learning paradigm in this model is more general than the learning paradigm in the classical model, even though its representations are somewhat restrictive.

In this chapter, we propose a model of learning systems that clearly defines the role played by the critic and the learning element, making explicit the knowledge they use during evaluation and editing (see Section 7). We also propose a generic scheme for training that makes it possible to acquire that knowledge automatically. Thus far, such knowledge has been programmed manually, making the resulting learning systems brittle when the training environment changes. Our model can adapt to changes in both the problem-solving objectives and the environmental reaction to the problem solver's actions.

1.6 Methods for Problem Solving

Problem solving models have become more general and powerful since the late 1960s. The first such model was proposed by Ernst and Newell [57] in their GPS program. More recently, the SOAR project [117] has put this model into a more uniform framework. In SOAR, all activity occurs as cycles of elaboration of memory, rule selection, and rule firing. Rules of preference encode all control knowledge, giving the system an ability to introspect and learn. Problem spaces and search schemes can be defined dynamically. Learning in SOAR is based on a simple, universal learning mechanism called chunking [115]. This system has been demonstrated in a wide variety of environments and on a variety of tasks: designing algorithms [222], simulating expert behavior [192], and simulating other learning mechanisms [193].

Simultaneous developments in parallel problem solving have led to the blackboard model of problem solving and learning [158]. Problem solving in this class of models (also known as BB* models) occurs through cooperative interaction among several parallel knowledge sources that communicate via a shared memory data structure called the *blackboard*. Recent extensions to the model [84] cover sources of control knowledge that make decisions at the strategic, policy, and scheduling levels, using a distinct control blackboard.

All the models described so far involve knowledge represented explicitly in a prescribed syntactic format. Connectionist problem solvers work differently [4, 11, 134, 198]. In these, problem entities are represented by a set of nodes. (Distributed, coarse-coded representations [87, 194] can provide robustness and generalization.) Connectionist systems model the search process by starting in an initial state and allowing the system to converge to some stable state [59, 88]. The long-term storage of relations among entities is encoded in the weights linking various units; the short-term state in the activation levels of individual units [196]. Typically, exploration of alternative solutions is made possible by introducing stochastic units [4, 11].

Connectionist problem solvers cannot easily represent quantifiers or predicate expressions of arbitrary complexity. One approach is to design increasingly complex ANSs for solving problems in variable binding and rule activation [50, 229, 230]. Our approach is to start with well-understood primitive networks (pattern classifier [197], autoassociator [3, 92], and bidirectional hetero-associative memory [111]) and to explore how a complex problem can be solved by an architecture composed of these basic units.

Heuristic knowledge in problem solving consists of associations between problem and solution entities. For example, if a connectionist network represents an association between the set of problem parameters and the set of operators, then it encodes heuristics for selection, ordering, and pruning of alternatives. Yet another network can learn a state evaluation function, thus supplying feedback to the action selection network [11, 12]. Note that this view of heuristic knowledge is consistent with the heuristic classification theory of problem solving [41]. Thus, we conjecture that the model used in this chapter will suffice for classification problems involving propositional variables. It has been shown by Clancey [40] that several decision problems are members of this class. For complex problems in planning and optimization, however, our results may not apply directly. In Section 5, we consider an important class of problems called dynamic decision problems, which give rise to some interesting problems in strategy learning.

1.7 Problem Representation

Choice of representation is an important determinant of problem solving performance [8, 9, 76, 98, 101]. However, a program must understand the role of representation to be able to adapt its representations automatically [127]. Because there has been no theory of representation in problem solving, it has been customary for designers to procedurally encode the knowledge of problem representation. Recently, however, researchers have tried to automate the process [66, 131, 151, 187, 188, 223]. This research stems from the work of Amarel [9, 10], who formalized the role of representation in problem solving; Davis [46], who used a methodology of metarepresentation in TEIRESIAS;

and Simon [211], whose UNDERSTAND program [209] was able to construct representations for problems stated in natural language.

Automatic acquisition of problem representation is beyond the scope of this chapter. Future developments may, however, significantly change the nature of strategy-learning systems. We consider the issue of representation complementary to the theme of this chapter.

Because we are experimenting with ANSs, a few remarks are in order regarding their powers of representation. The model assumed in this chapter is not as general as the problem space model of SOAR. We assume that ANS units collectively represent the set of problem parameters. A given network will apply to all instances of the same problem, but different networks will arise for different problems. It is possible to construct more general problem solvers by introducing an additional mapping network, but this will create complications because of unsolved problems in variable binding and unification. The simple model is sufficient to illustrate the utility of what we propose in this chapter.

There is another sense in which ANS-based models might be superior to conventional models. Conventional models assume fairly high-level primitive parameters and propositions that represent complex facts. There is little scope for partial or dynamically defined relationships between these facts. It is possible in multilayered ANSs both to construct abstract parameters of description in the internal hidden layers, and to represent partial relationships between primitive and abstract parameters [197]. This constructive element of ANSs lets them create complex descriptive parameters and form relationships between these and other variables of a problem [11].

1.8 Problem Solving Environment

A problem description covers only some of its instances. It does not include the environment of problem solving. Either the particular machine on which a problem is solved or the probability distribution of input variables may influence the optimality of a strategy. The *problem solving environment* consists of the world being modeled in the learning system, and the trainer (if there is one).

As an example of the influence that environment can have on learning, consider a computer with a two-level memory [253, 254]. It is possible that because of the high latency of accesses to the secondary memory, a depth-first search strategy (which has low memory requirement) might be preferable to a best-first search strategy (which has a high memory requirement). Without taking the memory latency into account, a best-first search may always seem preferable. The choice is obviously dependent on a context variable, namely, the amount of primary storage. In the rest of this chapter, we assume that the initial description is augmented with those context variables that affect the performance of a problem solver.

The environment plays an important role in strategy-learning systems because it reacts to the problem solver's actions. This reaction serves as feedback to guide future changes in the performance element. The nature of the feedback is an important characteristic of the environment. If the environment reacts immediately to every action, it has *immediate feedback*. Otherwise, it has *delayed feedback*, and the environment is said to be *slow reactive*. Bock [28] has suggested that feedback after a synthesized sequence of responses improves learning, when compared to systems with immediate feedback. The feedback takes the form of a desired response in a *supervised learning environment*, or of a periodic reinforcement in a *reinforcement learning environment*. In the former case, the feedback is said to be *corrective*, and in the latter, *evaluative*. In the case of delayed, evaluative feedback, the learning system needs an internal model of delay (hereafter called the *persistence model*). It uses the model to associate feedback with the actions causing that feedback.

Typically, the causal knowledge between actions and effects has an environment-dependent component. If this component is not known, the *causal model* of the system is incomplete. However, the causal model is important for proper attribution of feedback. Therefore, every learning system must have an internal model of its environment that captures the causal relationships between actions and effects, as well as the nature of the delays in the feedback process (in a persistence model).

An additional source of complexity in the organization of a strategy-learning system in a slow-reactive environment is the *asynchronous* behavior of the learning component. If the environment supplies inputs in the form of asynchronous events (which is typical of nonstationary dynamic environments), the problem solver must have an asynchronous architecture as well.

Finally, the environment controls the probability distributions of inputs. These in turn influence the applicability of various heuristic rules. This dependence can be easily represented by making the heuristic rules adaptive, and learning them through training in a specific environment.

Most of the existing learning systems have a fixed internal model of their environment. In Section 6, we examine the assumptions implicit in the design of the critic in several of these systems. In light of these assumptions, most existing systems work only in a limited range of situations. In particular, we show a class of strategy-learning problems that are hard to solve using the existing models.

1.9 Overview of This Chapter

In this chapter, we survey architectures for strategy learning. The survey is integrated with a classification of strategy-learning problems (Section 4) on the basis of several aspects of complexity introduced in Section 3. In Section 6, several architectures based on the Dietterich and Minsky models are examined. An improved model is proposed in Section 7. Unlike the Dietterich model, it does not rely on precise, complete, precoded knowledge of the environment; unlike the Minsky model, it can use whatever knowledge becomes available. It constructs dynamically an internal model of the environment. With this and other variations, our model is applicable to a certain class of problems that lies beyond the scope of both Dietterich and Minsky models.

The most crucial extensions of the classical model concern ill-defined objectives and delayed reinforcement. The classical model makes certain conceptual simplifications to eliminate these problems. We examine the deficiencies of several architectures based on this model by identifying a large class of problems that they cannot solve. We also study several systems based on Minsky's reinforcement learning model. These systems work well for small problems but require too many ad hoc decisions to work correctly on larger problems. We attempt to identify and explicitly represent the abstract bodies of knowledge acquired during learning. Our approach is illustrated with SMALL—a system that learns strategies for load balancing in multiprocessor systems [7, 24, 25]. As shown in Sections 4 and 5, load balancing belongs to a class of complex problems in our taxonomy.

2 THE NEED FOR STUDYING STRATEGY-LEARNING SYSTEMS

Even after almost two decades of prolific research in the design of intelligent machines, the applications have usually been demonstrated for simple domains. Attempts to construct complex real-world applications have been hampered by numerous factors, mainly attributed to the *knowledge acquisition bottleneck*. We examine several issues in knowledge acquisition that necessitate a reassessment of the available systems for strategy learning (Table 13.1).

Metaknowledge Not Available Strategy-learning systems have begun to require increasing amounts of metaknowledge from their designers. This knowledge is implicit in abstract parameters of representations and parameters of learning algorithms. The additional knowledge required to make a learning system work correctly has been called *bias*. The need for bias in learning systems has been recognized by several researchers (Watanabe [238, 239], Lenat [126], Davis [46], Utgoff [232, 233], Mitchell [143] and Rendell [184, 185]). These studies have quantified [183] the biases implicit in the choice of representations and in the meta-level architecture for rational setting of algorithm parameters. However, the critic has evaded similar analysis. Except for the work of Barto et al. [23], David Ackley [4], and P. J. Werbos [242], it has retained some very severe assumptions. A comprehensive treatment of the role of a critic in a learning system has been carried out by Dietterich and Buchanan [52]. Our approach is to make explicit the role of knowledge in credit assignment. Doing this enables us to analyze the critic at the knowledge level. Unlike most approaches based on the Dietterich model, we do not assume that the knowledge for credit assignment is available *a priori*.

The Need for Quick, Intelligent Decisions In real-time problem solving, strategic decisions must be made in limited time, using limited resources.

TABLE 13.1 Desiderata for Strategy Learning Systems.

Existing Systems	Proposed System
Algorithmic bias in the credit assignment process	Explicit representation of the knowledge required for doing credit assignment
Fixed strategies	Adaptive strategies
Abstract model of cost for learning and problem solving	Machine-specific model of cost for learning and problem solving
Assumption of stationary distributions	Adapting to time-varying distributions
Assumption of synchronized and/or immediate feedback	Ability to learn under asynchronous delayed feedback
TCA with implicit heuristics for persistence	Explicit, automatically-acquired model of persistence
SCA with fixed, user-supplied causal model	Dynamically varying causal model acquired automatically
Implementation on von Neuman architectures	Asynchronous, parallel connectionist architectures

The need to make quick decisions prohibits searching large search spaces, whereas the need to make intelligent decisions requires that good solutions must not be left unexplored. Moreover, the objectives of the problem may not be stated explicitly. These may have to be acquired or approximated by learning. Such constraints necessitate the inclusion of a learning component in any fast, intelligent problem solver. Learning and problem solving should be integrated so that they share processing power and other resources. Scheduling of resources is inevitable in these systems. The importance of a scheduler in a learning system was suggested by Wah and Yu [254] and in several blackboard control architectures [84, 158].

Studies of resource-constrained inference in AI have been restricted to considering some abstract performance parameters of measurement such as depth of chaining, number of states explored, the number of states memorized, the length of proposed solution, and/or the number of comparisons made. These parameters are meaningful only if they can be related to real measures of performance such as turnaround time on a task. Most researchers tend to ignore this issue because these relationships are machine-specific.

Consequently, most AI systems lack a machine-specific model of cost. Such knowledge can drastically change the evaluation of a strategy, because abstract measures of performance do not always produce a valid evaluation. An adaptive, explicit mechanism must be studied to acquire realistic performance parameters through training on the target machine. Our study in Section 7 includes the description of a system that addresses this problem. Our strategy-

learning system can, therefore, make use of machine-specific information in a structured fashion.

Nonstationary Dynamic Environment Many problems in the real world have too many parameters to be modeled exactly. Our focus is on computer architecture and performance evaluation. Traditionally, strategic problems in this area have been solved by approximation through stochastic modeling, but with the advent of large multiprocessor architectures, even approximate analytical modeling has become too cumbersome. Moreover, the environment in multiprocessor systems has been found to vary dynamically.

Recent experiments in the design of load-balancing strategies [7, 25, 163] have highlighted the need for dynamic decision making and, hence, for learning in real time. This, in turn, has posed the problems of resource-constrained and time-limited inference. Most existing learning systems do not have a real-time scheduling component and are therefore not applicable under these conditions.

Adaptive Strategies Strategic problem solving is needed when there is a large space of alternatives to explore and no mechanism is known for generating optimal or near-optimal solutions. Therefore, unlike earlier researchers (Slagle [214], Georgeff [72, 73] and Sacerdoti [201]), we do not assume that strategies are available *a priori*. This assumption must be dropped because of the need to make intelligent decisions quickly, and because of the lack of knowledge about the problem environment. Another reason is that computing environments change continuously, and it is impossible to estimate the goodness of a decision completely at any point in time. Thus, strategies should be adaptive in order to be really efficient and generally applicable.

Slow-Reactive Environments Learning is possible in the first place because the effect of every action becomes manifest in the state of the environment following that action. However, real environments are neither infinitely fast nor synchronous. The credit assignment process in most of the existing learning systems is based on the assumption of immediate feedback. Thus, feedback needs to be distributed only *structurally* among the various decisions causing an observed effect. In the case of delayed reinforcement, the credit assignment problem acquires an extra dimension: time! The feedback must be distributed *temporally* among several memorized decisions, before being distributed structurally among their individual causes.

Temporal Credit Assignment Temporal credit assignment (TCA) is the mechanism by which feedback resulting from several decisions is divided among the individual decisions constituting the episode. It entails associating observed effects with a history of actions. (See Sutton's thesis [225] for a comprehensive discussion of the origins and solution techniques of this problem.) The existing approaches to TCA mainly deal with persistence of effects caused by individual actions. They employ heuristics such as recency (that the more recent an action is, the more eligibile it is to receive feedback) and frequency (the more frequently an action occurs during the episode leading to feedback, the greater its eligibility).

Although we follow the example of Sutton [11, 23, 225] in separating the temporal and structural components of credit assignment, we relax the assumptions about persistence models. In particular, the model of strategy learning we propose to study has an explicit model of persistence and retains an explicit memory of past actions. In Section 7, we give details of a mechanism for dynamically learning persistence models of various actions and effects.

Structural Credit Assignment Structural credit assignment (SCA) is the mechanism by which feedback for an individual decision is divided among the rules causing that decision. SCA has been simplified in AI systems by assuming explicit representation of cause-effect relationships, as in the preconditions and postconditions of STRIPS-like operators. In nonstationary environments, this assumption leads to a loss of accuracy. We propose an architecture that dynamically acquires the causal model relating actions and effects.

Asynchrony in the Learning Environment In real-time systems, which tend to be event-driven, and in reinforcement learning systems, which tend to be driven by reinforcement, all behavior is inherently asynchronous. Therefore, the problem solver and the learner should be able to process asynchronous inputs and asynchronous feedback, respectively. Thus, our system must have an asynchronous architecture [84, 161]. We explore ANSs because asynchrony and concurrency are inherent traits of the connectionist model.

3 A TAXONOMY OF STRATEGY-LEARNING PROBLEMS

In this section, we study the factors that make some strategy-learning problems more difficult to solve than others. Our purpose is to identify the significant characteristics of the strategy-learning task and to show relationships among the problems studied by other researchers. Assuming that problems involving similar bodies of knowledge require similar knowledge acquisition techniques, we develop a knowledge-level specification of solutions to several classes of problems. Certain strategy-learning problems are complicated by characteristics that require that more knowledge be represented and reasoned with explicitly. These characteristics, along with the considerations discussed in Section 2, allow us to recognize those problems to which existing models of learning systems do not apply.

3.1 Nature of the Objective Function

The objective function of a problem is a generic description of the set of desired solutions. During credit assignment, the critic uses this description to assess

the quality of a solution. However, the specification of a problem might not include a crisp definition of its objective. We explore several possibilities for specification of problem objectives (see Table 13.2).

Single Built-In Objective The specification of the objective function is implicit in a credit assignment procedure provided by the user. Usually, a recognition procedure (as in Meta-DENDRAL [31]) is supplied for testing the extent to which a solution meets the criteria of a desired state. The strategy-learning task involves finding heuristics to generate candidate states, testing them with the given procedure, and generalizing these heuristics empirically. The generalization capabilities are implemented by the learning element. The credit assignment policy recommends the heuristics that generate acceptable states and censures those that do not. Consequently, the critic is simple to implement. This kind of system is useful in highly constrained environments where the specification of a desired solution is often too complex to be stated declaratively. However, such specification restricts the applicability of the learning system to just one problem domain. Also, some heuristic knowledge is embedded in the recognition procedure, and it cannot be modified easily because of the procedural encoding.

Assumed, General Objectives This class of learning scenarios are characterized by the *parametric* form of their recognition procedure. A classic example

TABLE 13.2 Effect of Objective Function Specification.

Type	Examples	Credit Assignment	Strategy Learning
Built-in single objective	Meta-DENDRAL	Sufficient prior knowledge for procedural implementation of credit assignment policy	Possible to use nonadaptive strategies
Assumed, general objective	Utility-driven classification	Credit assignment policy must be sensitive to parameters computed externally	Strategies should adapt to changes in external parameters
Flexible, explicit specification	Planning	Need to consider dynamic data structures representing causal information, such as goal hierarchies	Strategies must be declaratively represented and dynamically interpreted
Abstract, implicit specification	Reinforcement learning	Need to construct an internal model of the objective function; also need to acquire a causal model dynamically	Strategies must be adaptively defined; learning of causal models interleaved with strategy learning

of this class of problems is utility-driven classification, wherein the assumed objective is to minimize the probability of misclassification. Yet another example is MEA (means end analysis), in which the assumed goal is to minimize the difference between current state and the goal state. Learning in this case resembles an analytic optimization procedure [58]. The critic can be implemented procedurally. It computes an evaluation based on an objective built into the body of the procedure. Credit assignment involves the relative evaluation of states before and after each move, and assignment of credit based on the change in that evaluation.

Flexible Objectives, Explicit Goal Specification The objective in this case is defined dynamically. As a result, there can be no predefined procedure for evaluating an individual state. The critic must base its judgment on an acquired measure of goodness. The classic example of this line of problems is planning [200]. The first complication in these learning problems is that strategies have to be tagged with the preconditions of application. Each strategy is applicable to specific goals and specific problem situations. The critic tests the preconditions for overgenerality or overspecificity. It uses the causal model to identify the action(s) responsible for credit or blame. The learning element must understand the internal representations, and should be able to edit the preconditions and actions of a strategy.

Implicit Objectives, Abstract Goal Specification The definition of objective function is implicit in a periodic reinforcement signal. This class of learning situations is complicated because the trainer evaluates a complete solution path rather than evaluate each state on that path individually. The reinforcement can be interpreted as *the net change in a hypothetical static evaluation function* from the first state on the path to the last state. Learning involves starting out with unbiased random behavior, and then biasing it towards increasing reinforcement. The critic learns relationships between actions and subsequent reinforcement. It uses a causal model of the environment, but the model must be acquired automatically. The learning of the causal model is interleaved with the learning of strategies [23]. Examples include problems in reinforcement learning, such as learning to balance a pole in a cart-pole system [43, 139]. (A negative reinforcement signal is given every time the pole is so far off balance that it falls off.) In this example, the objective is to find a sequence of balancing actions that avoids a negative reinforcement signal for the maximum length of time.

A similar class of problems arises when objective function values are available for a sufficiently large set of problem situations but the form of this function is not known. The critic must induce an internal model of the objective function using these problem situations as examples.

Learning of the objective function can precede learning of strategies. An example is the *blackbox optimization* problem discussed by Ackley [4]. This case is more complicated than the case of parametric objective functions discussed earlier, because the parameters must be *learned* while searching.

In this last class of problems, structural credit assignment (Section 2) is more difficult because both the acquired causal model and the acquired heuristics are responsible for subsequent evaluation. A scheme must be devised so that credit or blame can be assigned to the component where the acquired knowledge is in error.

3.2 Immediate Feedback versus Delayed Feedback

Three classes of learning situations can be considered (Table 13.3):

Feedback after Every Decision In this class, the feedback from the environment is made available after every decision. Examples of learning from immediate feedback are strategies employing the *greedy heuristic*—choose the locally best decision at each decision point. However, a strategy for solving a problem that is not solvable directly (in one step) involves a sequence of decisions. Immediate feedback can improve sequential behavior only by improving each decision in isolation, assuming that the net effect on the global strategy will be favorable. There is no need for TCA because there is no ambiguity about the action responsible for the feedback. Several complex problems can be reduced to this class after temporal resolution of credit or blame [52, 120]. In real-world learning situations, immediate feedback is impractical because it slows down the problem solving system. Besides, real learning environments are not reactive enough to produce feedback after every decision.

Feedback after Every Solution Path These problems involve learning *episodes*; each episode consists of the complete solution of a specific training

TABLE 13.3 Effect of Feedback Latency.

Type of Feedback	Example	Credit Assignment	Strategy Learning
Immediate	Back-propagation [196]	Each decision examined in isolation	Possible to ignore the temporal structure of an episode
Synchronous, delayed	Systems based on the classical model [147]	Using prior information about the temporal structure of learning episodes	Knowledge of temporal structure must be supplied externally
Asynchronous, delayed	Reinforcement learning [23]	Need access to a persistence model for dynamic temporal resolution of feedback	Learning of a persistence model interleaved with strategy learning

instance and the feedback. TCA uses its knowledge of a solution to extract the temporal relationships implicit in that structure. In this case *synchronous delayed feedback** is available only after a complete solution path has been found. The critic has a procedural component for TCA. Abstractly, the decisions leading down the solution path are reinforced positively while those leading away from it are reinforced negatively. This type of feedback has been investigated by Langley [120] and Sleeman et al. [215]. The critic needs to memorize the solution path and the branches leading away from it. In some cases, TCA may involve additional search [145]. It is assumed, however, that the knowledge needed for TCA is made available to the learner before any learning takes place. Thus, even though these problems require some knowledge, such knowledge does not need to be learned.

Feedback at Arbitrary Intervals Most complex problems involve *asynchronous delayed feedback*. In these problems, there is no concept of a learning episode. A good approximation is to consider (imaginary) episodes delimited by successive reinforcements. In order to distribute the feedback the critic must memorize recent actions. However, the structure of episodic memory is not well defined. The memory consists of situation-action pairs along with the justification for each response. Because the critic has no knowledge of the internal structure of episodes, TCA is more difficult. Feedback is distributed among actions depending on their *eligibility* [139]. The eligibility of an action to receive feedback is computed as a function of the persistence of its effects. The more persistent the effects of an action are at the time of reinforcement, the greater is its eligibility to receive feedback. During learning, persistences of effects are maintained and associated with corresponding actions; these are updated in response to later actions, later events, and the passage of time. Thus, *the critic needs access to a persistence model, which maintains and updates persistences in response to events and actions, and which accounts for the decay of persistence with the passage of time*. The problem of learning strategies for pole balancing is an example of this type of feedback. In this problem, the feedback is available only when the pole falls off to one side. This feedback summarizes the evaluation of all the balancing actions since the last time such failure occurred. Moreover, this kind of failure cannot be predicted, given the knowledge of the problem.

3.3 Background Knowledge of the Environment

Background Knowledge Available We have already seen that some problems may require the knowledge of an internal model of the environment. In *knowledge-rich domains*, for example, expert systems and planners for finite worlds, one can assume complete, consistent knowledge. Learning in such domains involves operations like macro-operator formation and construction of censors and proposers for heuristic rules. Gaps in heuristic knowledge can be detected by inference based on background knowledge and evidence. The critic may incorporate a deductive component for explaining success and failure. When prior knowledge is not assumed to be consistent and/or complete, the critic should be capable of nonmonotonic inference.

* The word *synchronous* refers to the timing of the feedback relative to the timing of the decision-making process. Feedback is called *asynchronous* if one cannot predict its time of occurrence in terms of the latest decision that will be followed by feedback.

Background Knowledge Not Available In *knowledge-lean* domains, such as control problems for physical systems, there is little background knowledge. The internal model must be constructed by the learner through observation and experimentation. The critic may need to process stochastic data when either the distribution or some parameters of distribution are unknown. Learning in such domains involves constructing causal relationships based on the statistical analysis of dependence and the observation of temporal contiguity between actions and effects. The complexity of such analysis may be controlled by prior knowledge about a causal mechanism between certain actions and certain effects. Credit assignment prefers actions that are more likely to have such effects on the environment and are frequently associated with a better evaluation.

3.4 The Nature of Feedback

Abstract Advice If the feedback is in the form of abstract advice, then learning involves operationalizing this advice—rendering it usable for search by translating it into *concrete* advice. This may require knowledge for interpreting the abstract advice and repeatedly refining the interpretation until the advice can be stated in terms of the available heuristics. Examples include the FOO program [150]. This type of feedback requires that the trainer (advice giver) have perfect knowledge. Therefore, this type of feedback is not applicable when the environment varies dynamically.

Prescriptive Feedback If the feedback takes the form of a description of desired output, the critic computes error by matching. The credit assignment process assigns high evaluation to actions that minimize the mismatch. Examples of this kind of learning include systems based on inductive learning. The trainer needs to know the best answer (any good answer if the learning system tolerates noise) for specific instances drawn randomly from a population of problem situations.

Explanation If the feedback is an explanation of a solution in terms of recent behavior, the learner tries to generalize that explanation while maintaining consistency with other knowledge. Not only does the trainer need to know the correct response for randomly chosen problem situations, but also it needs to know *why* the response is correct. The learner, given an explanation in terms of its background knowledge, must isolate those components of knowledge that will be useful *in general*. Such techniques are applicable only to knowledge-rich domains.

Evaluative Feedback If the feedback takes the form of reward or punishment, the critic prefers actions that result in the maximum reward (minimum punishment). It needs to form an internal model of the reward generation mechanism. One approach is to translate the evaluative feedback into prescriptive feedback by learning an objective function that takes high values for positive evaluation and low values for negative evaluation [5]. This type of learning occurs in several dynamic control problems because they require minimum knowledge on the part of the trainer.

3.5 Strategy Selection versus Strategy Construction

A heuristic is modular, whereas a strategy is prescriptive [189]; this implies that heuristic-learning systems can consider decisions in isolation, but strategy-learning systems must deal with a decision in its context. Thus, heuristic learning systems are classified as strategy learners only in the case of delayed feedback. If they choose between several available heuristics, they are called *strategy selection systems*, whereas if they construct new heuristics, they are called (piecewise) *strategy constructors*. The knowledge used for credit assignment in each case is described next.

Strategy Selection Strategy selection problems are simpler than the corresponding construction problems. The selection problem requires knowledge of evaluation techniques only. If the learning problem involves evaluation of predefined operator sequences, then the critic needs to know only the extent to which the application of a sequence leads to a state that satisfies the objective.

Strategy Application If the problem involves learning of complex preconditions for assessing the applicability and the utility of predefined operator sequences, the credit assignment process must include an analysis of violated preconditions and their eventual rectification. Since the preconditions for individual operators are available, the role of each operator in a sequence may need to be assessed by the critic.

Strategy Construction A strategy construction problem requires knowledge of composition and evaluation of search control heuristics. If the learning task requires generation of complex sequential behavior, the critic needs the causal model and another component for constructing complex operator sequences. The latter component consists of operators for composition of operators and meta-level heuristics for the application of these composition operators.

3.6 Uncertain and Incomplete Information

If the problem knowledge is uncertain or incomplete, information must be weighted by some measure of belief, and beliefs must be revised based on evidence. Strategic knowledge in these problems depends on the state of the belief system, and needs to be revised in response to new evidence. The correctness of a strategy in the presence of uncertain parameters requires a statistical interpretation. The revision mechanism entails an explicit representation of the dependence between problem parameters, heuristic knowledge, and the net error due to uncertainty in either or both of these.

Uncertainty in Input Uncertainty in input may arise because of noisy, incomplete, or time-varying information. Noisy inputs force strategies to have some tolerance in the checking of preconditions. Incomplete inputs require that decision making should be possible even under partial information. If the problem includes time-varying parameters, belief in the latest measured values of those parameters should decrease with the time elapsed.

Uncertainty in Heuristic Knowledge Uncertainty in heuristic knowledge may arise due to either an inconsistent or an incomplete model of the environment. Strategies based on such knowledge have inherent uncertainty. The critic uses its error function and suggests modifications that minimize the expected error.

3.7 Resource and Time Constraints

Constraints on Object-Level Solutions Knowledge of resource and time constraints is made available to the critic so that it can reject strategies that violate them. In the presence of such constraints, the strategies must follow the principle of *bounded rationality* [208], which asserts that in real-world problems, there is a tradeoff between the optimality of a solution and the time spent discovering it. Optimality must be sacrificed sometimes. Any acceptable strategy must be capable of generating an answer within a predefined time.

Constraints on Learning The strategy-learning task should be completed in limited time. Learning algorithms that account for resource and time constraints require a substantial amount of scheduling knowledge and resource allocation heuristics. This knowledge may be implicit in the design of the problem solver and the strategy learner or may be made available in separate knowledge sources.

3.8 Types of Learning Techniques

A learning technique is said to be *supervised* if it involves a trainer and *unsupervised* if it does not. In between, there are techniques that reduce the role of the trainer; these are called *semisupervised*. Different learning techniques place different burdens on the learner and the trainer. In general, unsupervised learning is harder than supervised learning. Also, most of the supervised learning techniques rely on episodic learning, that is, learning under immediate or synchronous feedback.

Learning from Examples In the case of *inductive* learning of strategies from examples, the examples may be supplied externally along with the feedback. Such learning typically leads to problems in strategy selection. However, the critic's role is minimal because the evaluation is supplied externally.

Another form of supervised learning is called *explanation-based* learning. This type of learning is common in knowledge-rich domains. The critic has sufficient knowledge and inference machinery to construct a proof of the labeling (positive or negative) of the example. This proof serves as a causal model and is generalized by the learning element to extract the sufficient preconditions of the example solution. This technique constructs the causal model by inference on its domain theory. It needs explicit assumptions about persistence to work correctly in a time-varying environment [36].

Semisupervised Learning This type of learning uses the environmental feedback to generate more feedback internally. It is common in systems based on the Minsky model. It leads to problems of strategy construction.

Learning by experimentation is a semisupervised learning method. It typically uses a type of metaheuristic knowledge known as instance-perturbation heuristics, in order to generate examples internally. *Controlled experimentation* is a very powerful learning method in knowledge-lean domains as well as in knowledge-rich domains with inconsistent or incomplete theories [176-178]. The critic identifies candidate decisions whose ambiguity might be reduced by further experiments. It uses an explicitly represented causal model in order to detect such ambiguities. The type of feedback used is synchronous and prescriptive.

Learning in the presence of a critic (also called *reinforcement learning*) relies on evaluative feedback. The critic translates an evaluative measure into a prescriptive one by incorporating a prediction mechanism for reinforcement, which is refined after every prediction, based on a prescriptive measure: the error of prediction. The critic learns to predict reinforcement and induces a change in the direction of maximum reinforcement. The critic needs access to a causal model (in order to choose eligible candidates) and a persistence model (for temporal distribution of feedback).

Unsupervised Learning Unsupervised learning in the form of *learning by doing* is a useful way of minimizing reliance on an external trainer. This mechanism leads to integration of learning and problem solving. However, it may not be sufficient by itself because of the training time required. The critic must be able to create and evaluate hypotheses for explanation of unexpected

success or failure. The learning element integrates accepted hypotheses with the metaknowledge of the problem.

Another form of unsupervised learning, known as *self-supervised* learning, involves extra knowledge about *when* to learn. An example of this type of learning is *apprenticeship learning*, wherein learning occurs solely as a result of expectation failures.

Learning by Analogy Analogy is a general form of learning. It can be supervised or unsupervised, depending on how the strategies are generated initially. The problem solver has an explicit memory of solution schemata seen or derived in the past. It employs a reminding mechanism to retrieve a past instance in response to a new problem instance. The concept of a learning episode is crucial to this technique. If the storage and retrieval unit for the learning episodes is too large, then the memory requirements will grow and generalization abilities will drop, leading in the extreme case to a *memo-function* implementation of the problem solver. On the other hand, if the unit is too small, the retrieved solution will not have enough information to justify the overheads of a reminding mechanism. The critic's role is to detect (by matching) when a past strategy may be applicable. The learning element makes changes to the recalled strategy based on the match detected by the critic.

4 COMPLEXITY CLASSES FOR STRATEGY LEARNING

In the previous section, we saw how the complexity of strategy learning depends on some features of the problem class and the problem solving environment. Among the most important features are the nature of the objective function and the delay between an action and resulting feedback. In this section, we study several problem classes and relate them with various learning environments. This classification presents an integrated view of some well-known classes and well-researched learning environments (See Table 13.4). It allows us to assess the complexity of a strategy-learning problem in a given environment. It is also useful in studying the scope of existing learning models in terms of the problems and environments supported. We will introduce a class of complex learning problems in Section 5. The extensions to the classical learning models entailed by this class are studied in Section 7. For now, we focus on how various aspects of complexity are combined for some well-known problem classes.

As a general rule, problems requiring strategy construction subsume the corresponding problems of strategy selection; problems of learning under evaluative feedback subsume the corresponding problems of learning under prescriptive feedback; learning under resource constraints is usually more complex than learning without any constraints, and may require a radically different approach to learning and problem solving.

TABLE 13.4 Strategy-Learning Problems.

Problem Class	Simple Aspects	Difficult Aspects	Complexity of Learning
Learning strategies for combinatorial search	Well-defined objective, immediate prescriptive feedback	Large search spaces	Time-intensive knowledge-lean learning
Learning real-time scheduling	Strategy-selection, few alternatives	Complex objectives implicit in evaluative feedback, resource constraints	Dynamic learning of objective function and persistence models, only need knowledge for strategy selection
Traditional learning-to-plan problems	Background knowledge available, explicit objectives, immediate feedback	Dynamically defined objectives, delayed feedback	Knowledge-intensive, time-intensive learning; needs knowledge for strategy construction
Reactive plan revision	Explicit objectives	Dynamically defined objectives, asynchronous, delayed feedback, incomplete knowledge, resource contraints	Knowledge-intensive, time-intensive learning; needs knowledge for strategy construction; requires nonmonotonic temporal reasoning, persistence models supplied externally
Adaptive control problems	Strategy selection	Partial knowledge of complex nonlinear mappings, uncertain information, asynchronous feedback	Knowledge-lean learning; requires reasoning with uncertainty for learning nonlinear associations between time-varying, real-valued quantities

Uncertain information entails noisy or partial descriptions. It may not necessarily complicate learning, because uncertainty is inherent in learning and prediction. Hypotheses for explaining evidence naturally undergo revision in the learning process. It is possible to have a more elegant, uniform learning mechanism if data and inferences are allowed to be uncertain. Even though uncertain inputs and heuristic knowledge may require an expanded representation, they could actually reduce the complexity of learning and problem solving.

Among several possible training paradigms, learning with a critic under asynchronous delayed feedback requires the maximum knowledge. The critic has memory of several decisions not necessarily belonging to the same solution path, and it must isolate those that could have caused the feedback. Reinforcement learning with synchronous delayed feedback is less complex, because the critic has complete knowledge of temporal relationships within the sequence. As a result, the persistence model is considerably simplified. The critic does not need to resolve causal dependency between memorized actions and the feedback because all actions on the solution path are candidates for receiving credit or blame. Supervised learning with immediate feedback is substantially less complex because the learner does not need an internal model of persistence; it processes one decision at a time.

Learning Strategies for Combinatorial Search Problems Problems with well-defined objective function and immediate, prescriptive feedback are among the easiest learning problems in terms of metaheuristic knowledge.* This class includes combinatorial decision problems and combinatorial optimization problems. A decision problem involves finding any solution satisfying the problem constraints, whereas an optimization problem requires finding the best solution. Consequently, optimization problems are harder than the corresponding decision problems [69].

Real-Time Scheduling Problems Problems of learning while scheduling in real time are complicated by the presence of resource constraints. The objective function is often too complex to be stated explicitly. Instead, these problems are solved by making the feedback evaluative, so that an improvement in resource usage without significant loss of optimality is reinforced. Feedback is generated (either externally in case of supervised or reinforcement learning, or internally in the case of self-supervised learning) by taking into account the resources used by the proposed solution, as well as the extent to which scheduling constraints have been satisfied.

Planning Problems Problems in operational planning are characterized by the presence of complex logical constraints on the solution. Plans are usually discovered by inference with a causal model. Planning problems change character from constructive to selective if past planning episodes are stored and generalized. In the simplest case, an abstract plan is made available to the planner for operationalization by repeated refinement. This is a case of feedback by abstract advice. Learning in these situations has a strong deductive element and needs substantial background knowledge. Learning problems vary in complexity from those in which the background knowledge is assumed complete and consistent to those in which both these assumptions are dropped. Inconsistent knowledge requires default reasoning and nonmonotonic revision of beliefs. Learning is easier if the trainer supplies an explanation for errors

* In terms of heuristic knowledge required, these may well be the hardest solvable problems [132].

than if the system needs to maintain multiple possible explanations and the assumptions under which each holds.

Reactive planning, or planning in real time with resource constraints, has been the focus of recent research. Problems of this type are the most complex planning problems, because they involve time-varying parameters. Learning to plan in these domains involves constructing persistence models of various parameters. These models must account for unplanned changes in persistence caused by asynchronous external events.

Control Problems Unlike planning problems, most problems in control involve a large number of numeric, time-varying parameters. Control problem complexity grows from the level of system identification to that of policy design. The problem of system identification roughly corresponds to supervised learning of a function from input-output behavior samples. Adaptive learning of control policy has several features of complex strategy-learning problems: uncertainty, delayed feedback, and strategy construction. Problems in control have traditionally been solved using analytical techniques. Adaptiveness is limited to changes in parameters of a general model (such as the transition probabilities in a Markov process). These analytical techniques, however, require some serious assumptions about the distribution of inputs.

It is not our goal to design a fully general system capable of solving all kinds of problems, but we would like to demonstrate a technique for solving some complex problems in strategy learning. Our focus throughout the rest of the chapter is on a class of problems called *dynamic decision problems*, which share several complexity traits with the most difficult problems in our taxonomy.

5 DYNAMIC DECISION PROBLEMS

In this section, we define characteristics, cite examples, and identify components of solution for dynamic decision problems. This category includes many practical instances of problems, stochastically modeled, especially those whose parameters are not stationary with time. Many problems that can be modeled approximately by queuing theory fall into this group. Problems in computer networks, such as flow control, routing control, and congestion control, are all based on anticipated information that may not be collected accurately. Stochastic distributions are used as an approximation. Solutions of these problems can be improved using the deterministic information acquired by learning algorithms. Many problems in computer performance evaluation, such as buffer management, disk head management, and scheduling, are modeled stochastically (Table 13.5).

Dynamic decision problems can be further subdivided into policy design problems and reactive planning problems. A *policy design problem* is characterized by the need for fast decisions to be made from a small group of stereotypical decisions. Decision making in this case is too frequent to allow time for searching among alternatives. In a typical scenario, any one of a few available alternatives is implemented, and performance measurement is possible only through statistics accumulated over many decision points. Examples in this class are problems of load balancing in distributed computer systems, page replacement for a virtual memory hierarchy, and prefetching for instruction level scheduling of a pipelined computer. In contrast, a *reactive planning problem* [74] involves the dynamic construction of goal-seeking behaviors in the face of a nonstationary environment, in which asynchronously generated events may violate some goal conditions achieved by past actions. Such problems are common in robotics and assembly line planning. They require more effort on the part of the problem-solver/learner because no efficient strategies are defined *a priori*, and it is not possible to completely predict the environment in which the plan is going to execute.

Since the strategies to be developed are different in each subclass, different methods may have to be applied to learn the strategies. Examples of reactive planning arise in domains in which there are multiple autonomous problem solvers and none of them has a complete internal model of the other. Further examples include the open-ended cooperative planning problem of Konolige [107], and the blocks-and-baby problem of Schoppers [205].

TABLE 13.5 Strategy Learning for Dynamic Decision Problems.

Dynamic Decision Problem	Corresponding Strategy-Learning Problem
Load-balancing problem • Send jobs from computers with high workload to computers with low work load • Find a method to communicate information	• Finding attributes to adaptively characterize jobs, workload, and network traffic • Learn a model to evaluate effects of balancing decisions on response time • Find decision rules for reducing the response time
Page replacement problem • Replace the page that is not going to be referenced for the longest time	• Adaptively predict page usage • Find attributes on which to base predictions • Find decision rules based on predicted usage
Instruction scheduling in pipelined computers • Prefetch the instructions for the branch that is most likely to be the target of the next conditional jump • Find a schedule that keeps the pipeline full	• Adaptively predict page usage • Find attributes on which to base predictions • Find decision rules based on branch predictions

5.1 Characteristics of Dynamic Decision Problems

Dynamic decision problems may possess a combination of the following characteristics.

Nondeterminism and Dynamic Decision Making The decisions involved in solving these problems may require dynamic run-time information and information about characteristics of the problem instance being solved. Attempts at these problems are known to benefit substantially from the use of adaptive strategies. The best strategy for solving the problem instance cannot always be found *a priori*.

Multiple Strategies Typically, there is a large (and often intractable) number of alternative ways for getting a "good" solution. None of these ways can be selected *a priori* as the best one, because identification of the best strategy may depend on the problem being solved as well as on the values of certain time-varying parameters of the particular instance (possibly some environmental parameters). The enumeration of possible strategies to solve a problem under all possible conditions is generally infeasible.

Incomplete and Uncertain Information Complete information needed for making an accurate decision may be unavailable because either the overhead of collecting this information is too high or the information is heuristic and uncertain. The solution process should be able to accommodate incomplete and uncertain information and maximize the use of information as it becomes available.

Resource Constraints In a practical system, the available time and physical resources are limited. It is desirable that the best (as defined by the evaluation criteria) answer or strategy be found as quickly as possible. The large search space of candidate solutions and strategies prohibits an exhaustive search of all possibilities. An intelligent assignment of resources, both time and physical, must be made.

Delayed Evaluation The exact effect of making a decision may not be known until many decision points later. This requires a history of information to be maintained in order to predict the effect of a strategy. This problem of TCA is more difficult than is usually the case in current machine learning research, primarily because of the asynchrony in delayed feedback.

Dynamic decision problems include the problem of learning strategies for real-time scheduling and reactive planning discussed in Section 4. Certain complex problems in nonlinear adaptive control, such as the pole-balancing problem, are also members of this class. Because of the presence of resource constraints, this class does not include learning of strategies for optimization problems. However, for decision problems, the approach taken here is more realistic than traditional approaches such as decision theory (which considers each decision in isolation), or artificial intelligence (which usually assumes substantial prior knowledge). Also, the formulation of control problems here does not require prior knowledge of parameter value distributions. In recent past, some of these problems have been addressed by research in the area of ANSs. However, we focus on explicit consideration of delayed feedback and the role played by knowledge of the problem environment in credit assignment.

5.2 An Example of Dynamic Decision Problems

Dynamic decisions are almost inevitable in real-world problem solving. As already mentioned, the problems in policy design require different problem solving techniques than the problems in reactive planning. We restrict our attention to the former class in order to illustrate the proposed model in some detail.

As an example, consider the problem of designing a load balancing strategy. This type of strategy makes decisions about the distribution of jobs to various sites on a local area network so that some user-acceptable combination of the following objectives is optimized: overall job throughput, maximum completion time, average completion time, total communication cost, and average utilization of processors [37, 38, 156, 157, 237]. The information that can be measured includes processor utilization, number of disk requests per unit time, number of local and remote jobs that are pending, and status of communication links. These metrics are sensed periodically and used at each decision point. The set of conditions for determining the action to take at decision points constitutes the decision policy of the system. A *load balancing strategy* is, therefore, a combination of the metrics used and the decision policy [7].

A system for learning load balancing strategies should adapt to changes in its environment. This means that new combinations of low-level policy decisions and revised preconditions for applying various strategies, based on revised expressions of the measurable metrics, should be tried. Any changes should lead towards the desired goal of the system. When jobs are perfectly balanced, the interprocessor communication is at a minimum, and the overall throughput is high.

This problem belongs to the class of policy design problems. It involves run-time decision making; multiple strategies are available for solving the problem; the strategy-learning process must be executable within the time and resource constraints of the multiprocessing system; the information or metrics for the workload may not be up to date or may be uncertain; and the effect of a particular strategy does not become known until many decision points later.

We return to this problem in Section 7 following a survey of past research and a discussion of the new model of strategy-learning systems. This problem is also the subject of a prototype implementation [235] under development at the University of Illinois. However, it is only a representative problem. In general, other members of the problem class will have different objectives, decision variables, constraints, and entities. For instance, the three problems shown in Table 13.5 have similar knowledge-level learning requirements. All three have (1) a temporal aspect, which is manifest in the associated prediction problems; (2) a causal aspect, which is manifest in the construction of relevant attributes; and (3) the basic strategy selection aspect, which is manifest in the need for learning decision rules.

The methodology for the design of strategy-learning systems proposed in this chapter is motivated not so much by the specific constraints of any particular problem, but instead by the general characteristics of the entire problem class. Various aspects of complexity (Section 3) and our focus on the defining traits of dynamic decision problems (Section 5.1) underlie our survey of past research (Section 6).

6 SURVEY OF STRATEGY-LEARNING SYSTEMS

It is possible to classify learning systems along many dimensions. Three such dimensions were introduced by Michalski et al. [137]: (1) the role played by the learner (and the trainer, if there is one) and the amount of inference required; (2) the representation of acquired knowledge; and, (3) the domain of application of the performance system. Classification scheme (1) may be useful because the mechanisms of a specific learning model can be explained more concisely by examining the role of each component in a learning system, and by identifying the assumptions on which its design is based. Systems belonging to the same category under scheme (1) will employ similar mechanisms of acquisition. In strategy-learning systems, the representation of strategic knowledge exhibits substantial uniformity within a domain. Schemes (2) and (3) are equally good. The classification in this section employs scheme (3) at the top two levels and (1) at the bottom two* (See Figure 13.4)

The learning systems studied in the past have originated in domains as diverse as cognitive science, psychological decision theory, statistical decision theory, automated classifier systems, expert systems, and parallel distributed processing (or connectionism). It is interesting to note that the bodies of knowledge involved are very similar at an abstract level. Our view is that these systems are the implementations of one of the two learning models we have already introduced. When we identify a learning system or a class of learning systems as implementations of a model, we expect that the basic assessment of the model will carry over. We will, however, be careful to point out exceptions where appropriate.

Figure 13.4 shows the broad categories of learning paradigms we have studied. The hierarchy in the figure follows the general pattern

* Recall that the knowledge-level nomenclature introduced in Section 1.4 classifies learning paradigms primarily on the basis of the abstract acquisition mechanism employed. Specific systems specialize the abstract mechanism into an implementation scheme that is highly dependent on the representation used. Because there are fewer abstract acquisition mechanisms than representation schemes, we choose to classify by problem domains first.

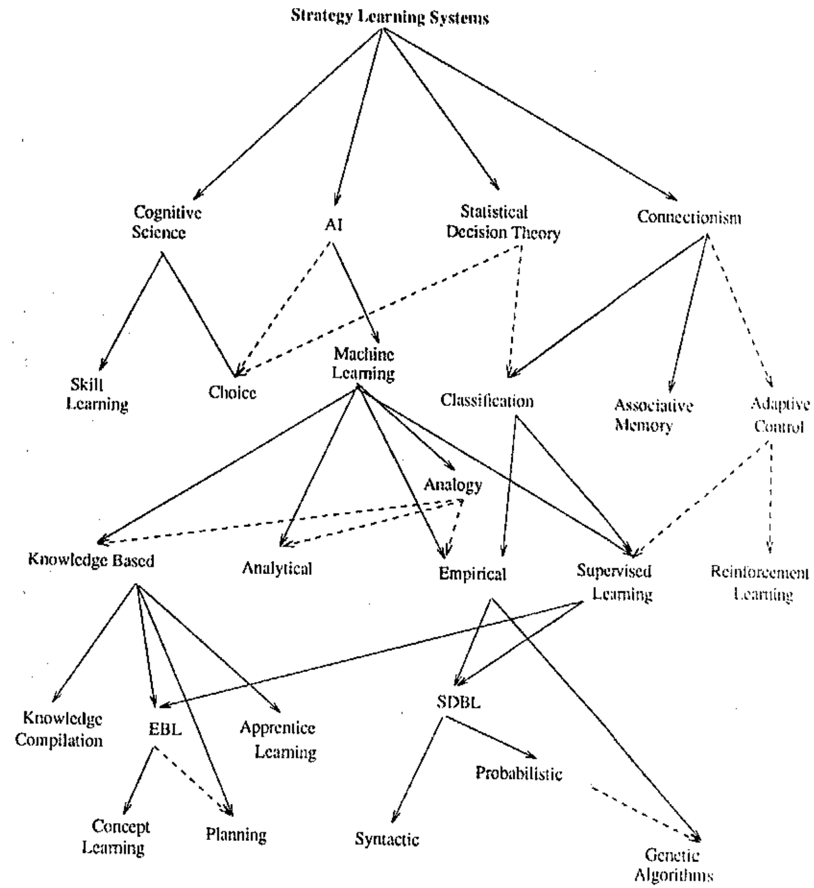

Figure 13.4 Strategy-learning paradigms

$$\text{domain} \rightarrow \text{problem} \rightarrow \text{extensions}.$$

The dotted lines in the figure represent approximate or indirect relationships. Some branches in the tree are short either because there is not enough variation in the abstract category, or because we are not aware of more than a few examples. In any case, this survey is meant to be representative rather than exhaustive. A quick look at the figure will, however, reveal that the strategy-learning problem has a vast scope and, therefore, a large number of techniques are available for solving it. We examine some of these techniques next and discuss their capabilities in terms of the aspects of complexity introduced earlier.

6.1 Cognitive Models of Skill Learning

The first models of problem solving behavior were motivated by a desire to understand human problem solving. Newell *et al.* [154] laid the foundation for a theory of intelligent problem solving by proposing the problem space model. The information processing theory of human problem solving [210] paved the way for later developments on the role of learning in intelligent behavior [118]. It provides common ground for comparing several strategy-learning paradigms. Even though the initial theory was built around the GPS program [57], which had a fixed general strategy of problem solving based on means-end analysis, subsequent refinements of the theory were increasingly oriented towards a production system model of learning.

The development of a cognitive theory of learning to solve problems was motivated by the search for general principles of learning, which would remain invariant across learning situations. Six ingredients of all learning situations were identified by Langley and Simon [118]: (1) *feedback* about the improvement or degradation of the performance element, (2) *behavior generation*, that is, the ability to explore alternative strategies, (3) a mechanism for *credit/blame assignment*, (4) the ability to *memorize* past performance and to *reassess* its goodness in the light of subsequent reinforcement, (5) the ability to *acquire* the examples, learning algorithms, and the causal model, by being told, and, finally, (6) continued *practice* of acquired skills. Depending on the nature of (5) above, the model can be classified as a variant of Dietterich model (if feedback is always told), a Minskian model (if feedback must be extrapolated from recent instances of being told), or neither (if the causal model and the persistence models are learned by means other than being told). These criteria show that the abstract view of learning is the same in cognitive science and artificial intelligence. Only the motivations differ. For an insightful treatment of motivation for learning, see the discussion by Simon [213].

In 1979, Anzai and Simon [17] proposed a theory of learning by doing that was centered around an adaptive production system [240]. That theory has been successfully demonstrated to acquire strategies [18] for a complex problem (steering a ship). The basic hypothesis of this theory is that problem solving proceeds by generation of problem spaces and search for goals in them. The theory applies whether or not the problem solver is capable of generating an initial problem space. In case the initial problem is ill defined, learning by doing can include understanding by doing. For example, it is possible to acquire the causal knowledge between actions and effects dynamically.

According to the theory, the basic unit of knowledge acquisition in strategy learning is the pruning heuristic, which allows the problem solver to ignore some parts of the search space during search. (For a computational approach to this kind of learning, the reader can refer to Yu's thesis [253].) The strategy-learning program is implemented as a rule discovery system much like Lenat's AM [124, 125] program, which employs heuristics for the discovery of heuristics. Learning by doing is capable of handling time lags between action and effect as well as problems with ill-defined objective functions. The theory, when augmented with the understanding-by-doing model, is capable of causal attribution and persistence modeling. Its reliance on domain knowledge and rules of discovery are a handicap in terms of speed. On the positive side, the theory is very general and applies across a variety of domains. The implementations of this theory lie outside the scope of Dietterich's model because of the understanding-by-doing component, and outside the scope of Minsky's model because of the ability to deal with metaknowledge for strategy acquisition. If implemented in full generality, following Simon's suggestion of employing recursively applicable heuristic rules for discovery, this theory can learn to solve any problem that is solvable by the production system representation. However, recent results in computational learning theory suggest that such a mechanism may not be able to learn within polynomial time [228]. In fact, such criticism applies to several other strategy-learning paradigms that are equally general (for instance, SOAR [116, 117] and ACT [14, 15, 152]).

There are several cognitive models of strategy-learning systems based on adaptive production systems with hardwired learning mechanisms in the form of procedures for composition of rules by chunking and proceduralization. These include learning by composition [13, 15], learning by knowledge compilation, the work of Lewis [129], and learning by chunking [191]. Although these systems are capable of constructive experimentation, they suffer from reliance on deep causal models of the problem domain, and the assumption of immediate feedback (for instance, p. 203 of reference [15]).

Still another class of cognitive models of strategy-learning systems are capable of learning problem solving skills from examples. In these systems, the dependence on background knowledge is minimized. The nature of feedback is prescriptive. Examples of this paradigm include the PET system for learning problem solving heuristics [103, 172, 174, 173] and the ALEX system for learning to solve textbook-algebra problems [153]. In a system based on ACT theory, Lewis and Anderson [130] consider the effect of delay on performance of human subjects. Their analysis seems to suggest that the knowledge needed for TCA is crucial to human learning, although no model of learning under delayed feedback is proposed. Most of these systems (except Anzai's learning by doing [18]) have been demonstrated on problems having a few completely specified parameters, an exact objective function, and immediate feedback for learning.

6.2 Strategy-Learning Systems for Making Choices

These systems have their origins in the psychology of decision making. Even though both psychological and statistical decision-making theories revolve around the concept of maximization of expected utility of the outcome by cost-benefit analysis, there is a major difference in their view of the learning process: statistical decision making is *normative* (how people *should* choose), whereas the psychological model is *descriptive* (how people *do* choose) [2]. Unlike the models of skill acquisition discussed above, these models take into account the uncertainty inherent in making strategic decisions. The major difference between the two models, insofar as their treatment of uncertainty is concerned, is their interpretation of probability. Psychological models use a nonfrequentistic interpretation, whereas statistical models use a frequentistic interpretation [86]. The psychological decision theory models restrict the information processing model by imposing the constraint of bounded rationality [212]. As a result, these models account for time and resource constraints. However, the temporal component of strategic behavior is played down; decisions are restricted to choices and preferences between options. Thus, these models are usually more appropriate for strategy selection and acquisition of guidance heuristics.

Einhorn and Hogarth [55, 56] identified cases in which feedback can lead to suboptimal performance. This is roughly equivalent to having no causal models. They also comment on how uncontrolled time-varying parameters make learning difficult. Both of their criticisms can be countered if a system is able to learn causal and persistence models dynamically. If there is *some* regularity in the variation of environment parameters, then the learner should be able to acquire a model of such variations by observation and controlled experimentation. While the researchers in psychological decision theory and in the psychology of learning [35] have both realized that the problem of delayed reinforcement is harder than the one with immediate feedback, there has been no attempt to construct a model that explains the temporal apportionment of feedback in delayed reinforcement learning.

Psychological models of choice under uncertainty are a rich source of possible representations of heuristic knowledge, providing models of heuristics that are invoked under resource constraints, and taking into account important variables of problem environment (see, for example, references [97, 99, 164, 199, 231, 26]). Their limited applicability in a computational environment is partly due to a lack of machinery for temporal credit assignment, and partly due to their extreme dependence on a knowledge-rich learning mechanism.

6.3 Strategy-Learning Methods in Statistical Decision Theory

As already mentioned, decision-theoretic mechanisms ignore the temporal aspects of strategies. They focus on the search for alternative means of achieving a *set* objective by ordering the alternatives according to some preference criterion [162]. In contrast, AI problem solving is concerned with problems that *cannot* be solved directly. Sequential behavior and explicit consideration of temporal aspects are central to AI problem solving. Thus, even though learning mechanisms in statistical decision theory are founded on sound learning rules, the major handicap of these approaches lies in their representation — only direct decision rules can be acquired. Decision-theoretic approaches have found their way into AI, in the form of mechanisms for assignment, fusion, and propagation of beliefs during uncertain inference, and for learning rules of selection.

The two major contenders for the use of decision-theoretic techniques in AI are the Bayesian nets of Pearl [168] and the reasoning systems based on the Dempster-Shafer theory of evidence combination [79]. These approaches are useful in learning from examples. However, the convergence and optimality of the rules depend on careful analyses of independence among the parameters. Especially, the Dempster-Shafer calculus is an excellent framework for performing resource-constrained inference [81], and can become an integral part of the performance element in a learning system. Numerous examples of AI systems using utility theory for assessing the value of a problem solving decision can be found in the literature [62, 67, 93, 195, 63].

Another class of limited strategy selection problems that can be solved using statistical decision theory techniques involve decision trees. These models assume that the decision tree is given *a priori* and that probabilities and utility values of leaf nodes are known. These techniques can tolerate some imprecision in the initial assignment of prior probabilities and utility values, and produce an optimal strategy, represented as a path in the decision tree [245].

The CART methodology of Breiman et al. [30] introduces mechanisms for automatic construction of classification trees from a learning sample. These mechanisms are nearly as general as conventional classifiers, and together with regression trees, they provide an efficient framework for managing dynamically varying representation of classes. They can be used in an adaptive action-selection network (performance element + learning element) in a data-intensive domain. Although the decision tree framework is more general than the single-decision model of choice, it requires complex mechanisms for propagation of belief. Because credit assignment can be viewed as revision of belief, these approaches can be applied to problems in which feedback is provided immediately after the last decision (leaf node) of a solution path is executed (synchronous delayed feedback).

We have criticized the decision-theory model for its lack of temporal structure, but there are a few exceptions. For instance, Kuipers [113] presents a strategy for dividing a complex (strategic) decision into a sequence of choices. Pollack et al.[171] have provided an overall architecture for integrating planning, belief revision, and choice. Decision-theoretic techniques are, therefore, promising as a model for data-intensive learning for strategy selection. Mechanisms that use explicit causal models in the form of dependency graphs (such as Pearl's Bayesian nets) appear useful for problems not involving delayed feedback. However, their neglect of temporal structure renders them unsuitable for dynamic decision problems. Recent advances in connectionist learning techniques of Sutton [227] and Werbos [241] have extended these models to include a mechanism for temporal prediction. In theory, all connectionist research can be interpreted statistically [77]. Therefore, statistical networks could be used for solving dynamic decision problems involving possibly nonlinear relationships between numerous time-varying parameters.

6.4 Strategy-Learning Methods in Artificial Intelligence

In AI, strategy-learning systems have been built for applications such as board games, problem solving, planning, and scheduling. Approaches have fallen into four broad classes:

1. *Empirical learning techniques* rely largely on syntactic similarity of examples in order to form general concepts. This approach can be further classified into (*a*) *similarity/difference-based learning* (SDBL), which works by generalization of logic expressions and structural descriptions given as examples of the concept; (*b*) *probabilistic SDBL*, which operates by forming regions in feature space through splitting and merging; and (*c*) *genetic learning* and other learning by experimentation methods, which use perturbation techniques to generate examples automatically.

2. *Analytical learning techniques* are based on extensive analysis of problem solving behavior with respect to one particular representation and a specific problem solving method.

3. *Knowledge-based approach* is used for acquisition of inference strategies in knowledge-rich domains such as expert systems, planning, and problem solving. Four learning techniques that belong to this class are: (*a*) *compilation*, which translates declarative heuristic knowledge into procedural solutions embodying the heuristics; (*b*) *explanation-based learning* (EBL), which acquires a general strategy from an explanation of a particular episode; (*c*) *plan revision*, which constructs and refines abstract descriptions of solutions; and (*d*) *apprentice* learning techniques, which employ learning by watching to fill gaps in the knowledge base.

4. *Analogical learning techniques* can be used in conjunction with any of the other learning techniques. Similarity between a new problem and an old problem, for which a solution is in memory, is exploited in order to systematically transform the recalled solution into a solution to the new problem.

In this section, we describe these learning methods in detail. Section 6.4.5 covers hybrid learning systems, which cannot be placed into any one class. Interested readers are also referred to survey papers exclusively dedicated to AI methods for strategy acquisition, by Keller [102], Mitchell [145], Sridharan and Bresina [219], and Langley [120].

6.4.1 *Empirical Learning Techniques.* Empirical methods rely on similarity/difference-based learning techniques to acquire heuristics and strategies. The primary categories within this class are systems that learn from examples and those that learn by experimentation. The critic in the first category of systems needs only know how to interpret examples; in the second category, however, it should be able to generate its own examples. Mitchell [145, 147] discusses a *version-space* approach for learning heuristics. The LEX system incorporates a flexible representation for partially learned heuristics, in which two boundary elements describe the least and most generally consistent heuristics with respect to the examples seen. The system can use learning by experimentation to restrict the version space. Its TCA employs time-bounded search, and the performance measure used is the processing time, which gives the system a machine-specific model of cost. However, the system relies on synchronous feedback. It uses procedurally encoded knowledge of the causal structure of the problem-solving episode. In dynamic decision problems, episodes do not have a predefined structure. LEX is based on the classical model of learning systems and therefore cannot be applied to problems involving asynchronous feedback.

The SAGE.2 [119] system's mechanism for credit assignment is flexible for both immediate and delayed feedback. It incorporates a framework for proposing rules for strategy learning. However, the system is very conservative in learning from examples. When a move is labeled undesirable, the system does not generate a negative example unless there is another move from the same state that has not been labeled as undesirable.

Another program that learns strategies from examples is ALEX [153], a nonfeedback learning system that relies on a form of precondition analysis. It uses a means-end analysis-based learner that only trains for differences between specific initial and final states. Credit assignment is based on whether the state is closer to the goal after applying an operator. The system has a condition creator to construct preconditions both from those pairs in successive states that cannot be explained by the current set of heuristics, and from information about the context of problem solving. Each step in the example is examined in isolation.

ALEX is also capable of learning strategies while solving problems. It learns from failure by specializing operator schemata through difference-based learning. Examples of failed episodes are examined for differences with similar situations that worked. This idea originated in the near-miss concept of Winston's ARCH program [250] and has recently been formalized by Falkenhainer [60]. Another example of this kind of learning is the discrimination learning mechanism in SAGE.2 [119].

Yu and Wah have developed TEACHER-1, a system that learns, by experimentation, the dominance relations in combinatorial searches [254]. The system generates alternatives by examining different parts of a search tree and proposes dominance relations based on positive and negative examples found. Credit assignment is based on the processing time expended, and the number of dominance heuristics found in an allocated time quantum. Dominance heuristics are hypothesized using domain-independent and domain-dependent knowledge. Because combinatorial search problems are well defined, and positive and negative examples are easily verified in the problem domain, dominance relations as good as those obtained by theoreticians have been found for a number of search problems.

Syntactical logical expressions can be generalized using Michalski's AQ program [138] and other generalization techniques [169]. These techniques have been used in various rule induction programs, such as Meta-DENDRAL [31], Poker Player [240], and UPL [160], for inducing rules from specific episodes of chaining inference. Generalization of structural descriptions can be used for learning macro-operators. Among several well-known programs for structure induction are ARCH [250] and INDUCE and its extensions [243]. Empirical techniques for learning macro-operators have been demonstrated by Whitehall [244] and Andrae [16]. Their systems generalize problem solving traces either generated internally or supplied by the user. The essential concept in this case is that of a problem solving episode. As we have already seen, in problems with asynchronous delayed feedback, the concept of an episode is very different, and so these methods do not apply directly. The same comment applies to other macro-operator learning systems, such as Macro Problem Solver [108] and FM [135].

Probabilistic methods employ numeric performance measures in making decisions. The PLS1 [180] system forms clusters of problem instances for which similar heuristics apply. Strategy learning is transformed into concept learning, and the objective function maps into the utility function of concept learning. The problem space is partitioned into rectangular regions by clustering techniques. These techniques are appropriate only when the problem-space parameters have smooth variations with respect to the applicable heuristics. (See, however, recent work on constructive induction [181, 186] as a possible counter argument.)

Zhang and Zhang [255] view search as a statistical sampling process. Learned strategies are encoded as weights on the nodes of the search space being explored. Evaluation functions of nodes that are unlikely to lead to solutions have weights added. Likewise, if a hypothesis is accepted, weights are added to all competing hypotheses. The likelihood of selection of a hypothesis varies inversely as its weight. The adjustment of weights on hypotheses amounts to learning.

Another class of search techniques use *dynamic weighting*. With heuristic estimates that are guaranteed lower bounds of the true goodness, branch-and-bound [114] methods can be used to implement learning while searching; even if the heuristics are not lower bounds, techniques such as the HPA dynamic weighting algorithm [170] and adaptive search [136] can be used to improve efficiency.

Besides being used to learn operator schemata and guidance rules, empirical methods have been also used for learning binary preference predicates [234], predicting the length of solutions [175], and for learning problem classes [19].

A class of empirical learning methods called *learning prediction* has much significance for learning in the Minskian model. The learner tries to learn a weighted evaluation function that remains invariant along the solution path, and that accurately predicts the assessment of the terminal node on the solution path [39]. The earliest example of such methods is Samuel's checker playing program [202] and its extensions [203]. Its equations have nearly the same form as the equations of the Minskian learning model in Figure 13.3. A uniform weighted average is used for maintaining the c_i's. The parameter N model is so set that oscillations are avoided during the early phases and overfitting is avoided later (N is 16 until the 32nd trial, 32 until the 64th trial, and so on, until it stabilizes at 256). Samuel's technique is quite ad hoc. Variants of this technique formalize the exponential decay factor [225] and the iterative prediction procedure. For instance, the Bayesian updating scheme is used by Lee and Mahajan [123], and linear regression is used by Rosenblatt's perceptron [190] and by Korf [110].

In subsequent research on prediction by temporal difference methods, Sutton [227] has presented a class of parameterized procedures called $TD(\lambda)$. The λ parameter is the exponential decay factor set by the user. A similar procedure had earlier been implemented in the Adaptive Critic Element [23], which can solve strategy-learning problems in knowledge-lean domains, using reinforcement learning under asynchronous delayed feedback. It includes a *horizon* parameter that defines the relative importance of making a correct prediction in the immediately following time instance versus the importance of the exponentially weighted sum of all the future predictions ad infinitum. These methods resolve the question of temporal credit assignment in a single-action machine by making the assumption that the eligibility of an action to receive feedback decreases exponentially with the time elapsed since the action was performed.

Genetic algorithms simulate nature's evolutionary mechanism to learn rules in propositional classifier systems. The classifier rules together constitute the performance element of these learning systems. This framework [90] has been augmented with learning based on genetic algorithms for discovery of new classifiers [29]. A novel TCA process called the bucket-brigade algorithm [91, 249] has been extended so that credit can be distributed structurally and temporally, using a synchronous delayed feedback paradigm of learning.

Each rule has an associated value called *strength*, and another parameter called its *specificity*, which is the fraction of the propositional literal population referred to in its antecedent. During problem solving, the *bid* of a rule is directly proportional to its strength and specificity. Among the eligible bidders (i.e., those whose preconditions are satisfied), some high bidders are selected for action. Credit is allocated either when external feedback is received from the environment, or when a rule causes another rule to be activated. Credit takes the form of increments proportional to the strength of the recommended rule. Sutton [227] notes that this algorithm results in exponentially decaying feedback, although this has not been established analytically. This algorithm is selective about the actions chosen to receive feedback, in some sense using the causal model implicit in the rules of the classifier system. It treats TCA and SCA uniformly. The eligibility of an action to receive feedback varies exponentially as its depth in the causal chain relative to the rule that draws the reinforcement. Under the assumption of infinite parallelism, the persistence model of this TCA policy is the same as that of Sutton's $TD(\lambda)$ family of procedures.

The genetic learning model is easily extended to environments with asynchronous delayed feedback [29], although the question of memorization of past decisions has not been addressed explicitly, and it is also a problem that the model's persistence model is a procedural component of the critic. This paradigm assumes a fixed objective function and a precoded causal model, although the causal model can be augmented by application of genetic operators, such as crossover, inversion, and mutation. These genetic operators are the strategy modifiers discussed in Section 1.4. The metaheuristics for the application of genetic operators are procedurally encoded in the critic.

Classifier systems share a lot of representational inadequacies with ANSs. Among them are restrictions concerning the number of propositional variables (which remains fixed throughout the learning and performance periods) and the inability to represent quantified predicates. The metaknowledge (the heuristics for classification problem solving) is expressed declaratively, and the learning technique is quite general.

Other examples of genetic learning systems for strategy learning are LS-1 [217] and PLS2 [179, 182].

6.4.2 *Analytical Learning Techniques.* Analytical methods are based on extensive analysis with respect to one particular problem representation and a specific problem solving technique. A typical example of this class is the DGBS system [58]. It is geared towards problems and operator representations specific to the General Problem Solver (GPS). It might be intractable on larger problems because it ends up examining all instances of an operator initially, instead of manipulating parameterized descriptions of operators. Strategy learning is reduced to automatic construction of triangular connection tables for GPS. The method takes as inputs the invariants of the procedure implementing an operator, and outputs difference-reducing connection tables. Two specific principles, namely, that the hardest difference must be reduced first, and that differences should not be reintroduced, are programmed into the strategy-learning mechanism.

An extension of DGBS [96] has been used for automatically acquiring heuristics for a robot planning problem. This method analytically derives subgoal-ordering heuristics and uses them on relaxed subproblems to get a heuristic estimate for the A* search procedure.

Other methods in this class include the heuristic generation method of Dechter and Pearl [48] for constraint satisfaction problems, the state-table analysis technique [112] for automatic completion of partial operator sequences, and the problem-relaxation technique of Gaschnig [70]. The major disadvantage of this type of method is the explicit dependence on restrictive representations. They assume that strategy construction meta-metaknowledge is understood so well that it can be implemented procedurally.

6.4.3 *Knowledge-Based Methods.* Knowledge-based methods are used for reasoning with explicitly represented knowledge. A strategy-learning mechanism in these environments can either result in the acquisition of new control (meta)knowledge, which can be subsequently used for controlling inference, or it can result in the re-representation of prior control knowledge so that inference becomes more efficient. The second set of methods are referred to as knowledge compilation.

Knowledge Compilation in Inference Systems This class of methods gives a general model for improving performance with practice. These methods apply well to problems with declarative representation of heuristic knowledge. The aim of the process of *operationalization* is to reformulate this knowledge so that the resulting procedures embody the heuristic knowledge and are directly stated in terms of the problem operators.

Operationalization may be applicable in two strategy-learning scenarios: abstractly stated strategic information to be translated into solutions, and abstractly stated objective functions to be translated into methods for achieving them. Note that learning must occur in a knowledge-intensive environment, with feedback taking the form of either abstract advice or abstract specification of problem objectives.

The first set of situations occurs, for example, in rule-based systems in which strategies are stated declaratively. An example is the ACT system [14], which employs two mechanisms called composition and proceduralization. *Proceduralization* eliminates matching and retrieval from long-term memory by instantiating variables in a constrained fashion while simultaneously dropping the constraining clauses, thus reducing the amount of information needed in working memory. *Composition* works either by collapsing two productions that follow each other and eliminating the intermediate goal, or by collapsing the preconditions and actions of a number of sequenced productions into one macro-production. The original productions are not destroyed during compilation. Other examples of this technique can be found in Section 6.1. In these systems, the composition and proceduralization mechanisms are the strategy modifiers. Storage economy is maintained by empirical generalization, and validity is maintained by specialization through difference-based discrimination.

The second category of systems are called *advice-taking* systems. Mostow [150] provides a comprehensive treatment of operationalization, which refines heuristic advice expressed declaratively into a procedure that incorporates the advice. The strategy modifiers in this case consist of *reformulation operators* and metaheuristics. The metaheuristics prune those alternatives unlikely to lead to an operational solution. An example of this category of techniques is Dietterich's test incorporation theory [53]. The Incorporation Problem Solver (IPS) is a solution construction process that transforms declaratively specified tests and naive generators into constrained, procedural, intelligent generators with little or no testing. The IPS is itself a problem solver employing test incorporation operators like serialization of subgoals (called *seed growth*), and non-repetitive enumeration (called *triangle generation*, a generator implemented as a function with memory). The learning element in these systems knows the internals of the generator and performs test incorporation based on transformed heuristic advice.

Compilation techiques can be used for solving problems with abstract, implicit objective functions in knowledge-rich domains. Objectives arising out of the consideration for resource constraints usually take this form. For example, it is usually clear what can be measured (recall *metrics* in load balancing), and it is usually clear what needs to be optimized, at least abstractly (e.g., the average turnaround time of jobs in load balancing). However, the nature of the relationship between the abstract objective and the observable metrics is not clear. This theory assumes that such a relationship can be inferred by deduction. This is a useful way of incorporating a machine-specific model of cost into strategy-learning systems for knowledge-rich domains. Another system that uses compilation to operationalize abstract objectives is MetaLEX—a system for improving the performance of LEX. The metaknowledge is stated explicitly in two forms: target concept knowledge (i.e., the concept of a successful move) and performance-system knowledge. The latter includes an internalized model of the environment, an abstract description of objectives, and the performance-system internals. MetaLEX compiles this metaknowledge and memorizes those parts of the compiled description that are particularly difficult to extract from the abstract objective.

For dynamic decision problems, these mechanisms of learning cannot be applied, primarily because of a lack of sufficient background knowledge. The form of the relationship between certain objective functions and observable parameters can at best be approximated by stochastic modeling, which requires analytical, machine-specific knowledge.

Knowledge Compilation in Problem Solving Perhaps no discussion of knowledge compilation would be complete without a discussion of macro-operators. We have already seen some mechanisms such as proceduralization (in rule-based systems) and learning of procedures from traces (in empirical learning systems). Explanation-based generalization as a mechanism for macro-operator formation is discussed in the next section. In the STRIPS/MACROPS system for learning and problem solving [65], *triangle tables* map problem situations to sequences of operators. They are constructed by learning while searching: any plan constructed *ab initio* can be stored in a triangle table. Plans are generalized using a method similar to the EGGS algorithm [148]. All constants in a triangle table are replaced by variables. The (overgeneralized) precondition in this *lifted* table, which contains all uninstantiated operators, is constrained by repeating the support proof. Only *necessary* instantiations of variables occur.

A recent extension to this framework introduces hierarchy in its notion of action. The tables can be used to map situations into sequences of abstract actions, which can have further internal structure [159]. These extended tables can be used in a hierarchical, asynchronous and concurrent control architecture. Triangle tables are suitable only for primitive actions representable as STRIPS operators—using add-lists and delete-lists to represent their effects. Their applicability to dynamic decision problems is, therefore, restricted to reactive planning problems.

Another method for compilation of macro-operators while searching is due to Korf [108, 109]. It has been used for solving problems with nonserializable subgoals. In such problems the problem objective has interacting, conjunctive subgoals that cannot be achieved by a problem solver that tries to achieve them one at a time. Korf's technique uses *macro tables*, which are triangle tables compiled to learn mappings from the initial state of a variable to its final state. Each entry in the table is a macro that leaves intact (as an end result) all previously achieved subgoals, and achieves one more subgoal. It is a weak method, capable of learning all the relevant macro-operators in the same order of time as is required to solve the same problem without any heuristic knowledge. It can be used for eliminating search from finite-space, discrete-domain planning problems. In reactive planning, goals achieved previously may be affected by future changes in the problem environment. For policy design problems, this method has limited applicability because of the presence of time-varying parameters.

Explanation-Based Methods The strategy-learning methods in this class rely on extensive domain knowledge. They work by explaining why a particular search path leads to success or failure. This explanation is generalized (sometimes specialized), and a sufficient set of preconditions inferred under which the same line of reasoning will apply. Acquired strategies, stored as schemas, are applied whenever their preconditions are satisfied. Search can be substantially reduced as learning proceeds and strategies grow in numbers and specificity.

The LEX2 system [145] uses goal regression to explain the success of problem solving episodes, and generalizes the resulting explanations. Applications of operators along the optimal solution path are the positive examples, and those leading away from the optimal path are the negative examples.

The EGGS generalization procedure [49, 148] reduces the problem of generalizing the explanation to a unification problem—that of matching the corresponding variables from adjacent rules in the explanation. EGGS can be viewed as an optimization procedure. Its goal is to maximize the number of models of its causal chain. The constraints on this procedure require that (1) the structure of the causal chain is preserved, and (2) the general explanation so obtained is valid.

The PRODIGY system [142] employs strategic information in the form of metarules (selection, rejection, and preference) to guide its search. The approach is unique in that it learns from successful solutions, failures, and goal interactions. Moreover, it learns by recursively specializing proof trees instead of generalizing ground proofs, which is the usual approach in EBL. It learns the *weakest* preconditions in which the schema represented by the particular solution is applicable. It is also able to handle disjunctive concepts (alternate paths to the goal). Carbonell and Gil [33] have incorporated learning by experimentation into the PRODIGY system. Their system is able to learn missing preconditions and postconditions of search control rules, and uses reactive experimentation to repair its strategies.

A major problem with explanation-based methods is their reliance on extensive knowledge, which is not always available. A learning system should be designed so that it does not always rely on complete and perfect information, but it should be capable of using domain-specific information when available. Most of the EBL systems fit well into the classical model of learning systems for knowledge-intensive domains with synchronized delayed feedback.

Planning Methods Planning methods are useful when the problem objectives are not directly achievable. They are suitable for reasoning about action, anticipating the world resulting from the consequences of an action, and generating behaviors for achieving the goals. Plans are constructed from the basic operators of the problem and other plans. Three methods for construction of plans are sequencing, iteration, and recursion. The distinction between goals and preconditions is important. Goals are the conditions that the planner tries to achieve, whereas preconditions are used by the planner only for checking the applicability of a plan. Goals can be complex requirements like protecting certain conditions through the plan body, preventing certain others from becoming true, or maintaining seriality constraints on the actions of the plan. In the literature, two kinds of techniques have emerged: planning as theorem proving and planning as problem solving. The former approach, also known as the *deductive* approach [133], views planning as proving theorems about the goals (such as *there exists a sequence of actions applicable to the initial state that will achieve the goal state*). The *problem solving approach* [200] views planning as a search in the space of plans, wherein the choice points result from the large number of applicable operators.

The strategic level in a planning system tackles the problem of metaplanning [221]. It was evident from the failure of early planners to solve some problems in conjunctive subgoals (e.g., see Sussman's anomaly [224]) that planners should avoid overcommitment. Thus, one commonly found heuristic at the meta-level is the *strategy of least commitment*. This and similar heuristics can be used to limit the plan construction search space. The basic operators available to the planning process include subgoaling (setting up new goals to complete a partial plan), temporal extension (editing old plans by extending them in either direction), and specialization (by making abstract actions more concrete, typically by instantiating some variable).

The following general learning scenarios in traditional planning have been identified by Collins [42]: (1) *caching and generalization*, so that there is no need to replan in similar situations in the future; (2) *repairing failed plans*, that is, detection, characterization, and removal of failure by plan transformation; (3) *apprentice learning*, that is, noticing the *unplanned* satisfaction of certain goals during planning, possibly leading to the discovery of some new plans; and (4) *learning from observation*, wherein the planner follows causal chains resulting from environmental events or another agent's actions, and memorizes those that achieve any of its goals.

The ability to plan is inherently related to the ability to detect and construct causal links. Therefore, the causal model of the planner is the immediate target of learning mechanisms. Learning to plan involves modification of causal mechanisms that lead to an erroneous action. It can be called *strategy learning in planning*, because it creates or modifies the pieces of a strategy for plan construction.

Learning to plan in dynamic domains is still an open problem. Time-varying parameters require that the planner have a persistence model of various conditions, because conditions previously achieved may be violated in the future. Detection of failures in planning, and their attribution to causes, is more complex due to the added responsibility of TCA. The problem of maintaining persistence models for planning has recently received a lot of attention [47, 82, 149].

Planning approaches provide an excellent framework for TCA by introducing *explicit* persistence models. However, the theories of reasoning with persistence models also need heuristics in order to get accurate projections. Several such metaheuristics are based on the least commitment principle. Some examples are discussed in the following:

1. *Persistence circumscription* [100]. The approach is to define an explicit termination predicate Clip, and to assert the persistence of a fact f as follows.

$$\mathrm{Hold}(t, f) \rightarrow \mathrm{Hold}(t+1, f) \oplus \mathrm{Clip}(t+1, f)$$

The circumscription of Clip is then solved in the theory, which includes the causal rules, the persistence axiom stated above, and the chronicle of events known to have occurred. Among all the minimal models, the one that has the longest persistence for facts is preferred.

2. *Motivated actions* [149]. A motivated action is one that is in all models of a temporal reasoning theory instance. In this approach, the models with the fewest unmotivated actions are preferred. Heuristically, this approach assumes that no action occurs unless it *must*.

3. *Probabilistic Notions of Persistence* [47]. When reasoning about the persistence of fact P, let E_P be the set of events that makes P true, and E_{-P} the set that makes P false. Letting $f(t)$ equal the probability $p(< E_P, t >)$ and $g(t)$ the probability $p(< E_{-P}, t >)$, the persistence of P at time t can be defined as:

$$p(< P, t >) = \int_{-\infty}^{t} f(z) e^{-\lambda(t-Z)} \left[1 - \int_{Z}^{t} g(x)\,dx \right] dz$$

This model counts some events more than once. The probability of occurrence of an event increases exponentially after the occurrence of a supporting event, and decreases similarly after an interfering event.

4. *Evidential model of persistence* [47]. This model takes into account several factors influencing the truth of a fact P at time t. These include a *natural attrition factor*, such as exponential decay, and a *causal accretion factor*, such as effects due to any element of E_P or $E_{\neg P}$. Evidential reasoning uses the *maximum entropy principle* as a heuristic for deriving a least-commitment assignment of probabilities.

In short, planning systems explicitly represent their persistence and causal models. They use meta-level heuristic knowledge to reason with these models. It is not possible to directly apply this kind of reasoning to policy design problems, but the model is suitable for reactive planning problems. Because our survey has been limited to the study of planning systems for dynamic decision problems, we may have overlooked some otherwise important aspects of planning. An excellent overview of planning techniques is given by Georgeff [75].

Learning Apprentice Systems These systems do not require explicit training. Instead, they just watch their users interact with a knowledge-based system. The apprentice embodies certain expectations at the meta-level of the problem. Learning opportunities arise when an expectation fails due to a discrepancy between what the apprentice expects and what the user does. Such systems are also called *lazy generalizers* [174].

Apprentice learning can be combined with knowledge-based inference mechanisms in several interesting ways. The PROTOS system employs an exemplar-based categorization technique in which specific examples are retained, and these guide difference-based learning in the face of discrepancies. Another system employing this technique has recently been proposed by Wilkins [247]. The ODYSSEUS system learns by trying to complete explanations for failure of expectations. In the process of explaining a failure, it discovers gaps in its knowledge. New knowledge is hypothesized in order to fill this gap. The validity of the new knowledge is established by a confirmation theory built into the system. The apprentice learner of BB1 [83] learns new scheduling heuristics when its preferences are overridden by a human expert.

All these systems are *discrepancy-driven*. They exhibit learning with asynchronous immediate feedback in knowledge-rich domains, and derive their power from rich, explicit knowledge of abstract control metarules. They represent perhaps the most pragmatic of all knowledge-based learning paradigms, in spite of their complete reliance on generic knowledge sources that monitor the application of heuristic knowledge.

6.4.4 Analogy-Based Methods. Analogical methods rely on knowledge of solutions for problems already solved. As a generic paradigm, analogy is closely related to generalization. However, in the context of knowledge-based learning systems, analogy has come to acquire a rather specialized meaning.

The method is specifically interpreted as a mechanism for the recall of past solutions, based on the similarity between the new and the old problems, including transformation of old solutions for application in the new situation. An *analogical mapping* is derived by comparing the two situations and their respective contexts of application. Part of the old solution is transferred under this mapping to the new problem situation. Analogy-based learning is practical when there is sufficient previous experience in solving similar problems.

Analogy is a very general learning mechanism applicable to almost any problem solving and learning technique. Gaschnig [70] describes an example of *analytical* analogy, Winston [251] discusses empirical structural learning by analogy, Carbonell [32] discusses planning by analogy, Davies and Russell [45] describe a deductive approach to reasoning by analogy, and Greiner [80] proposes an analogy-based approach that can be easily combined with EBL.

6.4.5 Hybrid Methods. Recently, various researchers have attempted to reduce EBL's reliance on perfect domain theory. Pazzani [165] uses the similarity of explanations to induce generalized explanation schemas, thus devising a method for *explained* empirical classification. Flann and Dietterich have proposed another extension to standard EBL, an approach that can handle multiple examples. This approach is called IOE. It involves constructing individual explanations for each example, generalizing over explanations, pruning away dissimilar subproofs, and compiling the generalized explanation into a schema.

Star [220] has proposed a method of combining EBL, similarity-based learning, and decision theory. His method, called T-BIL (Theory-Based Inductive Learning), has been applied to a reactive planning problem in robotics. Feedback in this system is in the form of explanations. The technique can handle multiple examples: after the first time, all subsequent explanations are used for generating *observation reports*. These reports are then used to update the probabilities of causal rules using a Bayesian updating mechanism.

Mitchell [147] (who called this approach *analytical* learning) proposed a method for combining EBL with an empirical approach based on the version-space method. The version-space method maintains a partial description of a heuristic in the form of two propositional descriptions S and G, which correspond to the most specific and the most general instances in a propositional Boolean lattice of the set of examples seen so far. The S and G descriptions found by the empirical learning program LEX are analogous to the necessary and sufficient conditions for application of the heuristic, as derived by the analytical learning program LEX2. Thus, the conditions derived by LEX2 could be used as positive instances for generalizing S. The resulting hybrid method was applied to simplifying integration formulae.

The UNIMEM system [122] uses similarity-based learning to infer the plausibility of causal relationships before applying EBL. Danyluk [44] has also explored the idea of combining empirical and analytical learning techniques. His technique differs from UNIMEM in that structural descriptions are matched and that inexact matching is allowed.

The idea of applying EBL as a verification step has been proposed by Golding et al. [78]. Their system learns strategies from expert advice. It works by asking an expert for advice when it cannot proceed during a problem-solving episode. The advice is followed, and the steps leading to the solution are retained, verified, and generalized by chunking.

Other methods include the precondition analysis technique used by Silver [206] in his heuristic learning system LP, and a more general method suggested by Desimone [51]. Both methods are more general than SDBL because they analyze examples by reasoning about the purpose of each step, and less powerful than EBL because they consider only the preconditions of rules rather than their interactions [148]. Silver's approach can only learn a linear sequence of rules. Desimone extends his approach to nonlinear solution trees using dependency graphs. These methods have been demonstrated on high-school algebra problems.

Rajamoney [176, 177] uses a combination of EBL and learning by experimentation to refine an initially imperfect domain theory. His technique relies on a form of causal isolation known as factoring [71]. Controlled experiments are designed from the partial description of failed instance of strategy application. One of the causal mechanisms (or processes) is allowed to dominate each experiment. Factoring simplifies the design of the critic and the learning element. This technique is applicable to knowledge-based learning for control problems.

In Section 6.4.3, we have discussed the PRODIGY system of Minton et al. [142]. Not only does this system learn from positive examples of strategy application, it also learns from explanation of failure. This technique combines EBL with compression analysis [141], a knowledge compilation technique that performs a utility-guided search through a space of plausible explanations. It results in more effective explanations, whose operationalization results in more efficient strategies.

The methods discussed in this section demonstrate that no learning technique is sufficient by itself. While empirical learning techniques exploit syntactic and numeric similarity, explanation-based learning uses similarity of causal structure. Apprentice learning techniques are adept at patching knowledge gaps, and so are learning-by-experimentation methods. Knowledge compilation improves efficiency by priming patterns of inference, and planning techniques have the constructive element necessary for creation of new strategies. Effective learning must employ a hybrid of these methods. The spectrum of learning techniques in AI is the richest, most implementable, and also the most well researched of all domains. However, existing learning mechanisms must be combined in order to design more useful and more generally applicable techniques.

6.5 Connectionist Methods

Connectionist methods also address the problem of representation and acquisition of strategic metaknowledge, but the perspective here is different. The assumed underlying model of intelligence is a massively parallel problem-solving model. All computation emerges from the collective activity of a large number of simple and richly interconnected units. Such systems for problem solving and learning are variously known as *parallel distributed processing* (PDP) systems, *neural networks*, or *artificial neural systems* (ANSs). A comprehensive review of the paradigm can be found in the references [1, 196].

What are ANSs? A connectionist network (Figure 13.5) can be viewed as an active data structure consisting of *units* interconnected by weighted *links*. Some designated units act as sensors, so that input to the system can be supplied by influencing the states of these units. Other units act as effectors, so that the state of these units can be used to direct actions. The remaining units (appropriately called *knowledge atoms* by Smolensky [218]) capture the relationships among sensors, between sensors and effectors, and among effectors. The global short-term state of the system is captured locally by the activation of each unit. The interconnections among units allow interaction between states. The long-term memory of the system lies in weights on the links between units. Learning in these systems occurs by the modification of these weights.

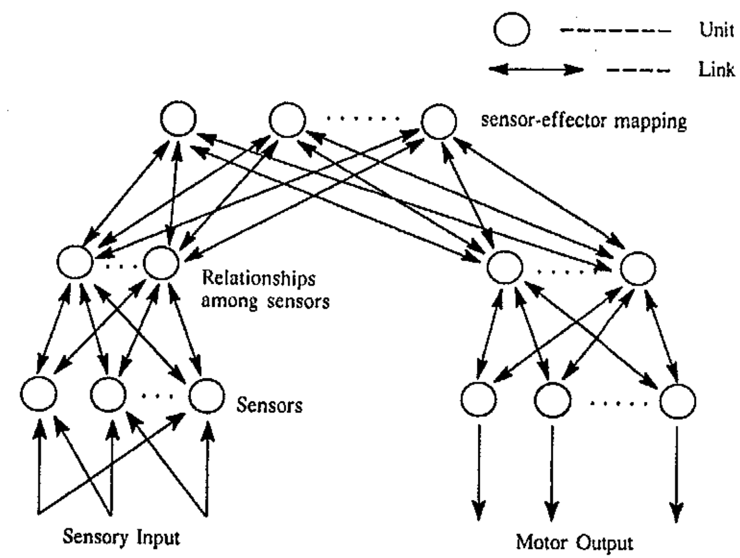

Figure 13.5 A typical connectionist network.

Connectionist Primitives Connectionist primitive is a generic term for the various types of knowledge that can be represented using ANSs. The following primitives are discussed and used in this chapter:

1. *Pattern classifier*, which is capable of judging whether or not a pattern of inputs is a member of a class;
2. *Pattern associator*, which learns (possibly bidirectional) associations between input and output patterns;
3. *Auto-associator*, which is capable of learning how to complete partial patterns of input, based on the memory of patterns seen in the past; and
4. *Competitive activation*, which uses competing units with mutual inhibitory connections, to select the most strongly activated unit.

Learning in ANSs A *connectionist learning paradigm* is a method for modifying weights (also called synaptic weights) in response to a pattern of changing activations. Among the several well-known paradigms for adjustment of weights with experience, we shall cite one for each kind of primitive unit discussed. Rumelhart et al. [197] have developed the *back-propagation* algorithm for learning in pattern classifiers; Kosko [111] has developed the Adaptive Bidirectional Associative Memory (ABAM) framework for learning pattern associations; Kohonen [105] has developed the Learning Vector Quantization (LVQ) framework for self-organizing auto-associative memories; and Grossberg [34] has developed the Adaptive Resonance Theory (ART2) framework for competitive learning.

ANSs for Strategy Learning Although only a few applications of ANSs to strategy learning exist, this approach has some distinct advantages over others. In particular, decision making under partial information is easier. Connectionist networks have spontaneous generalization that results in a transfer of expertise to problems with similar values for parameters of representation. They are ideal for storing nonlinear, time-varying associations between situations and actions. The disadvantage, on the other hand, is that all knowledge is implicit and cannot be translated into other representations. Therefore, there is a tradeoff between flexibility and performance between connectionist and conventional representations.

Anderson's thesis [11] presents a comprehensive treatment of search problems using connectionist neural networks. Two separate networks are used: one for learning the state-action mapping, and the other for learning the state-evaluation function. A partial solution to the TCA problem is presented in the form of a feed-forward connection between the input layer and the state-evaluation layer. This allows the system to plan at least one move ahead of its output. The pole-balancing problem [12] is used to demonstrate the applicability of modular neural networks to complex decision problems. The system learns discriminatory features of parameter space that lead to useful representations of inputs from the perspective of strategy selection.

There are some important points to be learned from this approach. It demonstrates the feasibility of neural networks for learning state-action mappings, and more importantly, for learning continuous-valued evaluation functions using a fixed set of parameters. This means that for certain strategy-learning problems in which numeric parameters abound and the evaluation function depends in a complex way on these parameters, it might be worthwhile to sacrifice explicit knowledge of problem objectives for greater ability to learn. For problems with ill-defined objective functions, the objective function can be acquired during the course of learning through external evaluation of certain result states. Connectionist systems might be the only resort for such problems, because none of the other systems work without an explicitly stated goal.

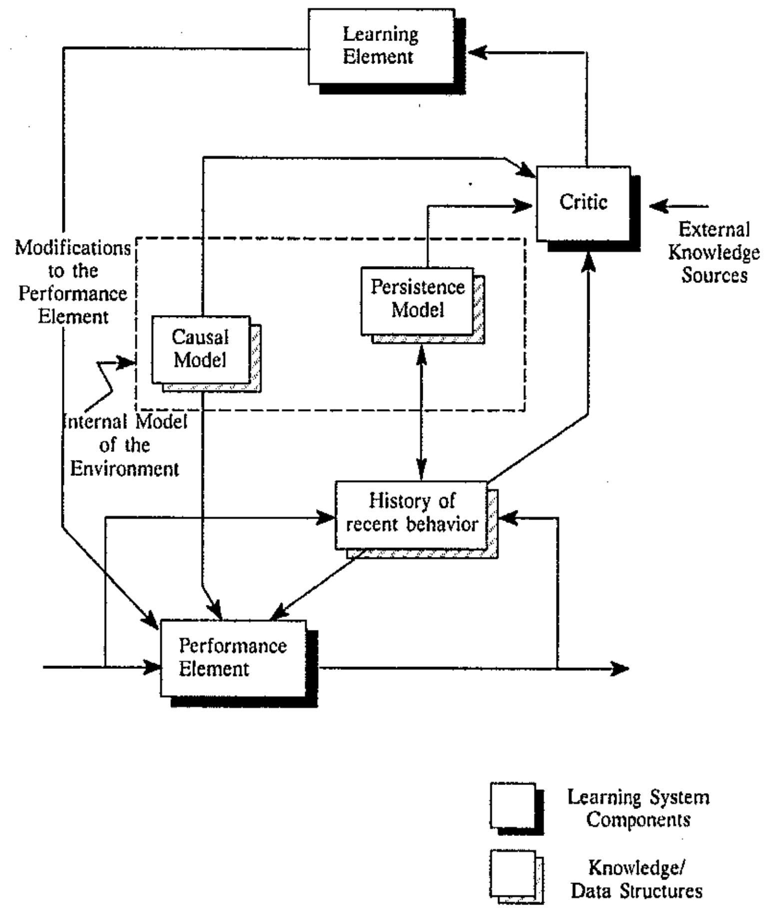

Figure 13.6 Proposed model of a learning system.

7 A PROPOSED MODEL OF LEARNING SYSTEMS

In this section we describe a learning model that solves complex problems in temporal credit assignment using explicit knowledge (see Figure 13.6). Instead of using the same network for storing past decisions, heuristics, and eligibilities of decisions, our model works with an explicit memory containing tuples that have the following abstract form:

$$< \text{problem instance, decision} >$$

The system is trained incrementally, in stages. In the first stage, a model of the reinforcer's objective function is acquired by repeatedly observing the reinforcement resulting from a single controlled action, and learning an association between the effects of that action and the feedback.* Learning in this stage involves relating an abstract specification of objectives with some observable parameters of the system. In a typical dynamic decision problem, there are so many observable parameters that (1) it is impossible to represent all the associations using just one network, and (2) it is not feasible to vary them simultaneously in a controlled fashion. Therefore, this stage is further divided into substages, each corresponding to the learning of a partial description in terms of a small group of parameters known to be mutually dependent. This is one way of using background knowledge in designing pattern classifiers with a large number of input parameters—letting the connectivity of the network reflect the top-down expectation of high covariance among related inputs. Several such partial associations are used to incrementally estimate the structure of a complete association. This scheme is appealing because it parallels the general pattern of evolution of complex, hierarchical systems [207]. Learning stays within resource and time bounds, and knowledge of complex associations can be accumulated incrementally.

In fact, this principle of layered information compression (term due to Rendell [181]) is also applied to the overall training phase. The remaining bodies of knowledge are acquired in subsequent stages. Each stage of learning a complex association is further subdivided by recursively applying the layering principle. The causal and persistence models are learned separately using controlled experimentation on the environment. The causal model is acquired first by varying the environmental control variables and performing random actions, one at a time, and acquiring associations of the form:

$$\text{Actions} \times \text{Perceived World} \rightarrow \text{Hypothesized World.}$$

The next stage in learning is to acquire the persistence model. Once again the control variables of the environment are varied over trials. First the environment is allowed to reach quiescence following a randomly chosen action. The action is performed and the change in the effect, as well as its average decay rate following the action, are observed. These are used to update the *causal accretion* and *natural attrition* components of the persistence model. The *spontaneous causation* component is acquired by probabilistically choosing an event, letting it occur, and observing the change in the particular effect under study. This stage results in the acquisition of associations having the form:

$$\text{Events} \times \text{Hypothesized World} \rightarrow \text{Hypothesized World,}$$

$$\text{Hypothesized World} \times \text{Time} \rightarrow \text{Hypothesized World,}$$

$$\text{Actions} \times \text{Hypothesized World} \rightarrow \text{Hypothesized World.}$$

The final part of the training phase involves experiments for learning search control heuristics. In this phase, the acquired objective function, the causal model, and the persistence model all remain fixed. External feedback can be supplied by comparing the heuristic performance against the best nonheuristic performance. Alternatively, some of the feedback can be generated internally using secondary reinforcement-learning algorithms. The feedback is apportioned, first temporally according to weights in the persistence model, and then structurally according to weights in the causal model. At the end of this phase, the performance element is also trained.

We have now begun to operationalize (to borrow a term from knowledge-based learning) our knowledge-level specification of the model. In this section, we have shown how the problem of learning can be reduced to a problem of acquiring complex associations. We have also proposed a *generic* layered information compression paradigm for training. Our description is still coarser than an implementation-level specification. This discussion, therefore, is applicable to almost all the implementation models of learning systems discussed in Section 6. In the following, we propose a connectionist implementation of our model, for solving a representative dynamic decision problem.

ANS-Based Implementation of the Proposed Model The implementation schematic shown in Figure 13.7 shows how the variety of associations that need to be acquired may be represented using connectionist primitives. Figure 13.8 shows the connectionist implementation of the critic. It differs from the secondary reinforcement-generation mechanism of Minskian models in that it has hooks for using explicitly represented causal and persistence models. It is different from the classical model because of its predictive reinforcement mechanism and the flexible structure of its episodic memory.

The performance element is implemented using a pattern classifier network. A competitive network is used for selecting an action given a problem instance. The causal model is realized in an ABAM (adaptive bidirectional associative memory). The persistence model and the memory of recent decisions are imple-

* This approach can be replaced by the Sutton-Barto model [23] for acquiring the weights of the adaptive critic element, which, with small modifications, can be used to acquire the objective function by performing multiple actions at a time. However, the heuristics implicit in that model will cause problems in learning.

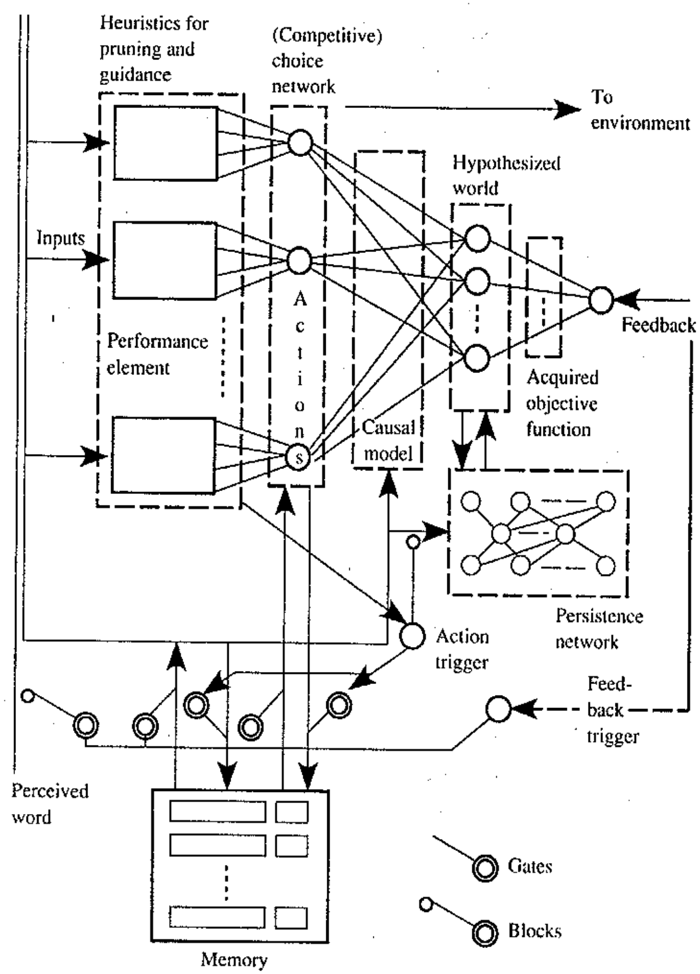

Figure 13.7 Schematic of ANS implementation.

mented using auto-associators. Numerous triggers and gates help synchronize the operation of the ANS ensemble.

Learning Strategies for Load-Balancing We view the load balancing problem as a search problem. Let migrate (t, i, j) denote the operator that migrates a task t from processor i to processor j. A balancing decision can then be seen as mapping the set of operators migrate (t, i, neighbor(i)) to processor i. The system makes migration decisions based on local workload information and task parameters.

The learning system for workload characterization and workload distribution is based on connectionist learning systems [89]. These have been successfully demonstrated to learn complex associations without being given any

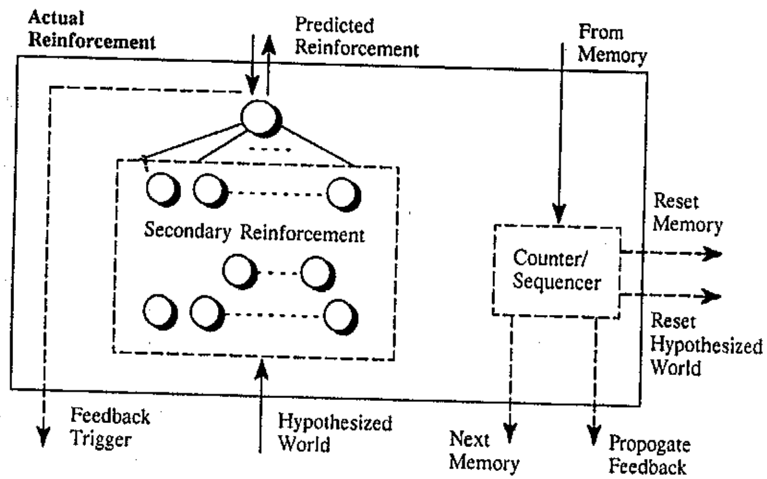

Figure 13.8 Schematic of the critic.

knowledge of the parametric form of the association. They construct abstract features of the input parameter space in their internal layers (also called hidden units). Learning and problem solving can be easily integrated in these systems, because learning proceeds by modification of weights in the problem solving component. It has been shown [23] that reinforcement-learning systems implemented as connectionist networks can withstand delays in feedback. Finally, the connectionist model of problem solving is inherently asynchronous and, once trained, can provide quick answers by parallel examination of various attributes making up the state.

The Learning System Two kinds of learning are addressed. The *search space learner* is a learning component implemented separately at each site in the distributed system that learns the workload characteristics of that site. These characteristics can be nonlinear functions of the available primitive measurements and inputs. The acquired workload parameters define an abstract search space. The *search heuristics learner* learns the relationship between workload information and the corresponding load balancing decisions. This component is essential because some of the attributes in the abstract search space are not directly available and their relationship with decision making changes dynamically.

The system operates in three distinct phases: (1) training for workload characterization, (2) training for workload distribution heuristics, and (3) application. Phases (1) and (2) involve learning.

Controlled experimentation is used for learning in phase (1). The response time of the system is observed for a wide variety of controlled tasks. For each experiment, the task remains fixed. Learning is further simplified by only learning a formula for the *relative* change in response time, rather than learning the response time function itself.

Learning in phase (2) is more complicated because a history of past actions must be maintained. Moreover, explicit justification must be maintained for each action. When external feedback is provided, it is distributed temporally and structurally. The information needed for doing credit assignment depends critically on the nature of feedback.

In our system, feedback is generated by comparing the improvement in response time for various decisions. The information for structural credit assignment is stored in the *causal model*, and that for temporal credit assignment in the *persistence model*. During training, the weights in the causal model are adjusted to reflect relationships between load balancing decisions and workloads. This is achieved by running controlled experiments—both the task and the load balancing action are control variables. The weights in the persistence model are acquired by controlled experimentation too. They represent the relationship between workload information and load balancing decisions on one side, and the time for which the effects of balancing persist on the other.

Controlled generation of problem situations in load balancing gives rise to other problems. One such problem is that of synthetic workload generation. We have developed synthetic generators for various computational resources—CPU, disk, and memory. Using these and an intelligent sampling technique, the learning program can sample the critical portions of the input parameter space.

When the inputs are gated into the heuristic networks, many action units may simultaneously try to go on. In this case, competition among these units results in the selection of some unit with strong backing from guidance heuristics and weak opposition from pruning heuristics. This action is used to update the persistence of effects it is known to cause. After this, if no action occurs, then the natural attrition component of the persistence model takes over. If we allow asynchronous external events to influence persistences, then a sudden change in environment, measured as a differential between the current state of the perceived world and the state immediately preceding, can also change the persistence of certain effects.

As soon as the competition between actions is resolved, the action trigger goes on, and the state of the perceived world and the chosen action are gated into the memory. This process continues until an asynchronous external feedback signal becomes available. The learner shuts its perception and focuses on learning rather than acting. The feedback is, meanwhile, backed up to the individual effects of the hypothesized world via the objective function model, where it stays till the end of the learning phase.

The learning phase occurs in cycles of recall and modification. The recall phase involves selecting a memory instance and gating the memorized inputs to the input lines. The network corresponding to the memorized action is selected for update. The persistence-weighted feedback is propagated back via the causal model of that action. The forward propagation of perceived inputs through the heuristic mechanism (but not the competition mechanism) is followed by a pass of the standard back-propagation method. This allows

the feedback to be distributed among the weights of the heuristic mechanism. This cycle repeats until all the memories have been exhausted or the time runs out for the learner.*

During phase (3), the weights in the action selection network are used for deciding the next action. The search for the actions proceeds asynchronously and concurrently as jobs originate. Conflicts are resolved by winner-take-all [64] networks. All the weights (in the causal and persistence models, and in the action selection network) remain fixed during this phase.

Load balancing is a difficult problem in parallel search. The problem relies greatly on statistical prediction of patterns inherent in the environment. However, any predictive solution in a stochastic environment is at best approximate. Formal proofs and determination of the accuracy of approximation will be carried out in the future.

The unique feature of our system as compared to past solutions on workload characterization and distribution is that strategies are not built in. This allows the system to adapt to installation-specific and architecture-dependent parameters of workload—an activity traditionally performed by human experts. The systems reported so far, such as Engineering Computer Network [94, 95], MACH [20], Locus [235], and University of California, Los Angeles's Locus [236] and System V, depend on user-defined workload metrics for making load balancing decisions. This is not only nonadaptive, but also not portable to systems of different configuration.

We have suggested an adaptive approach to designing a system for learning load balancing strategies. The top-down development illustrates our design methodology for strategy-learning systems. The basic steps can be summarized as follows.

1. Analysis of the *knowledge-level requirements* of the learning task;
2. Study of the *knowledge-level functionality* in available learning systems;
3. Systematic mapping of various bodies of knowledge to suitable learning mechanisms;
4. Principled design of a training scheme using layered information compression.

8 CONCLUSION

In this section, we summarize our opinions on various issues pertaining to the architecture of strategy-learning systems. These conclusions are based upon an extensive survey of architectures, covering diverse domains and a variety

* It is possible to be more intelligent about the whole learning process if we also maintain, in the history, persistence of each action. This value should be set to its maximum value (say, 1) at the time of gating an action into the memory. Each time one of the original effects of an action is updated, the persistence should be decremented by an amount proportional to the magnitude of the change.

of techniques. Perhaps the most important issues we have identified are the effects of delayed feedback and ill-defined objectives on the design of learning systems. Such learning situations are inherent to a large class of problems called dynamic decision problems. We have identified the bodies of knowledge involved in intelligent credit assignment for such problems and illustrated how such knowledge may be acquired automatically. We have attempted to advance the field of machine learning by indicating a general methodology for learning to acquire strategies in complex domains under difficult learning situations.

We end this chapter with observations central to the architecture of strategy learning systems for complex domains.

Learning in Complex Domains Perhaps the most important variable in real-world domains is time; it also happens to be the most neglected variable in research on learning strategies for solving problems. Considerations of delay had recently led to significant changes in learning strategy, as exhibited by Sutton's work. However, even though a few learning heuristics (such as recency and frequency) had been proposed and included in a learning algorithm, the *knowledge* for designing such heuristics for other complex problems was not explicitly available. The knowledge involved is analogous to persistence axioms being used in systems for temporal planning. This observation is being used to improve the design of learning systems by reducing the bias. This leads to systems that represent this knowledge explicitly and attempt to acquire it automatically for specific domains and/or learning environments. It also suggests a technique for implementation of temporal planning using connectionist systems similar to the ASE/ACE system studied by Barto et al. [11, 23, 227].

A Second Opinion on the Design of ANS Learning Algorithms The trend in the design of learning algorithms for connectionist systems has been to keep them simple and local. While these goals are the crux of the argument for connectionism, they have been misinterpreted by recent research as an argument for doing all the learning in their systems through intelligent, heuristic, local, and simple algorithms. This should remind us of the classic heuristic problem solver of Slagle [214] in the beginning days of AI. The next stage in AI was its most progressive one; and the one characteristic that separated the AI systems of this age from those of its early days was the *separation of knowledge from the algorithmically implemented inference component.* Is it not obvious that if connectionist systems are to implement anything like a general purpose learning substrate, they should explicitly represent as much knowledge as possible? The human learning system, which, for now, continues to guide these developments, works in slow learning environments, in the presence of distractions, in the presence of varying amounts of delay, and in situations where goals (objectives) are set dynamically in response to asynchronous, sporadic reinforcements. Realizing that any heuristic component of learning algorithms that is associated with a dynamically varying feature of the learning environment must be flexible and available for introspective modification, we can immediately form an argument against all the learning being done algorithmically.

Our model suggests how connectionist systems may continue to use simple algorithms and still remain sufficiently flexible to be able to learn in complex environments. The major idea of our methodology is to translate complex learning into phases of knowledge acquisition. So, what should be the next stage in the design of intelligent connectionist architectures? Our answer would be: the development of *modular knowledge-level architectures* for learning. Systems employing heuristic learning algorithms are important in their own right because they establish the need for certain kinds of knowledge. However, from an architectural point of view, they only illustrate the need for an additional *module* rather than a change in a *learning algorithm*. General learning in complex situations is achievable not through more complex learning algorithms, but through more advanced interactions among dynamically acquired bodies of knowledge.

If some bodies of knowledge cannot be acquired or represented by currently existing ANS primitives, that should motivate the search for new learning algorithms and primitives. Examples of such primitives have been set forth by Touretzky et al. [230] in the domain of connectionist rule-following systems [229]. It is our conjecture that future connectionist architectures will be modular, motivated by a knowledge-level analysis of the learning scenario, and that the search for new algorithmically implemented connectionist learning procedures will only be motivated by the lack of a connectionist substrate to implement one or more types of knowledge modules. This conjecture is consistent with the fundamental tenets of connectionism as well as AI.

Reassessment of the Design of Multiprocessor Operating Systems Our experience with distributed systems in the recent past has indicated a need for automation in the management of these systems. Traditionally, complex problems in the design of multiprocessor operating systems have been solved algorithmically. Many of these problems fit into the framework of dynamic decision making. The problem of learning a load balancing strategy is representative of this class. Adaptiveness in the design of these systems is an essential step towards complete automation. It is quite plausible to think of a neural network on every processor making complex scheduling or routing decisions in response to the changing operating environment. The advent of neural networks opens exciting avenues for designing faster, more efficient multiprocessors.

The research reported here can be viewed as a marriage between learning and architecture. New developments in architecture that might lead to fast, general implementation of connectionist primitives can be used for efficient, low-overhead learning, and new developments in the field of learning can lead to the design of intelligent, efficient multiprocessor architectures that will be spending less time controlling themselves. This view has begun to be shared by other researchers in the field [68, 121, 163].

Relationship with Psychological Models of Learning We have taken an engineering approach to the design of learning systems. Psychological models, on the other hand, describe how human beings behave and learn. The complex human learning system and its methodology can guide the engineering approach. Thus, one direction in which we hope that our survey of cognitive models will benefit learning research is clear. In the opposite direction, design issues in the architecture of learning systems might shape research in psychology by providing a model and vernacular for the description of human learning.

Learning under delayed reinforcement is not a well-understood topic in psychology. Indeed, as we have indicated earlier (Section 6.1), some psychological evidence points against the incidence of such learning during simple controlled episodes of solving simple problems. However, there are complex learning phenomena that occur over prolonged periods of time and cannot be tested in a laboratory setting. Indeed, Anzai's model for skill acquisition [18] includes delay models that have not been understood and explained in a problem-independent way.

In several psychological models there is an implicit assumption that learning occurs when the learner is *prepared* to learn. This preparedness has not been characterized well. It is our hypothesis that the acquisition of causal models and persistence models sets the stage for learning of problem-solving heuristics. It is unfortunate that because of the lack of proper terminology, Anzai [18] includes both kinds of knowledge in what he calls causal knowledge. We hope that our model will help further the productive exchange of ideas in these two disciplines by providing a common vocabulary.

Issues Not Addressed Several issues need to be resolved before general purpose connectionist systems can be constructed. Among the most important issues are scaling (How well does an architecture scale up to larger and more complex problems?) and generalization (How well does the system perform on problems it has not been trained on?). These issues are the subject of ongoing research [6]. Another issue is the design of network architecture. It is obvious that there is design bias implicit in the choice of a network architecture and size. Also, the choice of an encoding for continuous-valued or integer-valued parameters leads to representational bias. The choice of a learning mechanism that incorporates more than just the internal representation used by the performance element gives rise to algorithmic bias.

We have not identified any new connectionist primitives, although, as indicated by Touretzky et al., the existing set might be inadequate for modeling all intelligent behavior. Yet another topic that needs more attention is the design of a serializing component (*scheduler*, one might say) in order to map the large number of simultaneously active bodies of knowledge onto a constrained architecture, and to achieve this mapping and management at a small overhead. Finally, the symbolic interpretation of the activities of an ANS remains an open question, and more work is needed on that front.

9 ACKNOWLEDGMENTS

We express our thanks to the members of the staff at the RII division of NASA Ames Research Center for supporting our research activities in the summer of 1987 and 1988. Thanks are also due to Dr. Katherine Baumgartner for sharing the results of her extensive study on the resource allocation problems in distributed systems, and for providing the skeleton for the load balancing testbed; and to Albert Yu for interesting discussions leading to the formulation of dynamic decision problems. This research was supported by the National Science Foundation under grant MIP-88-10598, and by the National Aeronautics and Space Administration under grant NCC 2-481. This chapter has benefitted from critical comments of Mark Gooley, Larry Rendell, Chris Matheus, Peter Haddawy, Carl Kadie, and Munidar Singh.

REFERENCES

1. *DARPA Neural Network Study*, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA, July 1988.
2. R. P. Abelson and A. Levi, "Decision Making and Decision Theory," *Handbook of Social Psychology*, 1983.
3. D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, vol. 9, pp. 147–169, 1985.
4. D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing*, Kluwer Academic Publishers, Boston, 1987.
5. D. H. Ackley, "Reinforcement Learning with Back Propagation," *Abstracts of Neural Networks for Computing*, April 1988.
6. S. Ahmad, "Scaling and Generalization in Neural Networks: A Case Study," *Proc. 1988 Connectionist Models Summer School*, eds. D. Touretzky, G. E. Hinton, and T. J. Sejnowski, Morgan Kaufmann, Palo Alto, CA, 1988.
7. R. Alonso, "The Design of Load Balancing Strategies for Distributed Systems," *Future Directions in Computer Architecture and Software Workshop*, pp. 1–6, Seabrook Island, SC, May 5–7, 1986.
8. S. Amarel, J. S. Brown, B. G. Buchanan, P. Hart, C. Kulikowski, W. Martin, and H. Pople, "Report of Panel on Applications of Artificial Intelligence," *Proc. IJCAI-77*, pp. 994–1006, 1977.
9. S. Amarel, "On Representations of Problems of Reasoning about Actions," *Readings in Artificial Intelligence*, eds. B. L. Webber and N. Nilsson, pp. 2–22, Morgan Kaufmann, Los Altos, CA, 1981.
10. S. Amarel, "Program Synthesis as a Theory Formation Task: Problem Representations and Solution Methods," *Machine Learning: An Artificial Intelligence Approach*, eds. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Morgan Kaufmann, Los Altos, CA, 1986.
11. C. W. Anderson, "Learning and Problem Solving with Multilayer Connectionist Systems," Ph.D. thesis, University of Massachusetts, Amherst, MA, 1986.
12. C. W. Anderson, "Strategy Learning with Multilayer Connectionist Representations," *Proc. 4th Int'l Workshop on Machine Learning*, pp. 103–114, June 1987.

13. J. R. Anderson, J. G. Greeno, P. J. Kline, and D. M. Neves, "Acquisition of Problem-Solving Skill," *Cognitive Skills and Their Acquisition*, ed. J. R. Anderson, Lawrence Erlbaum Associates, 1981.

14. J. R. Anderson, "Knowledge Compilation: The General Learning Mechanism," *Machine Learning: An Artificial Intelligence Approach*, eds. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Morgan Kaufmann, Los Altos, CA, 1986.

15. J. R. Anderson, "Skill-Acquisition: Compilation of Weak-Method Problem Solutions," *Psychological Review*, vol. 94, pp. 192–210, 1987.

16. P. M Andrae, "Constraint Limited Generalization: Acquiring Procedures from Examples," *Proc. Nat'l Conf. on Artificial Intelligence*, pp. 6–10, 1984.

17. Y. Anzai and H. Simon, "The Theory of Learning by Doing," *Psychological Review*, vol. 36, pp. 124–140, 1979.

18. Y. Anzai, "Doing, Understanding, and Learning in Problem Solving," *Production System Models of Learning and Development*, ed. Klahr et al., MIT Press, 1987.

19. A. A. Araya, "Learning Problem Classes by Means of Experimentation and Generalization," *Proc. Nat'l Conf. on Artificial Intelligence*, pp. 11–15, 1984.

20. R. V. Baron et al., *MACH Kernel Interface Manual*, Pittsburgh, PA, January 1987.

21. A. Barr and E. A. Feigenbaum, William Kaufmann, Los Altos, CA, 1981, 1982.

22. A. G. Barto and R. S. Sutton, "Landmark Learning: An Illustration of Associative Search," *Biological Cybernetics*, vol. 42, pp. 1–8, 1981.

23. A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike Adaptive Elements that Can Solve Difficult Learning Control Problems," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. SMC-13, pp. 834–846, 1983.

24. K. M. Baumgartner and B. W. Wah, "Load Balancing Protocols on a Local Computer System with a Multiaccess Bus," *Proc. Int'l Conf. on Parallel Processing*, pp. 851–858, University Park, PA, August 1987.

25. K. M. Baumgartner, *Resource Allocation on Distributed Computer Systems*, West Lafayette, IN, May 1988.

26. L. R. Beach and T. R. Mitchell, "A Contingency Model for the Selection of Decision Strategies," *Academy of Management Review*, vol. 3, pp. 439–449, 1978.

27. H. J. Berliner, "Some Necessary Conditions for a Master Chess Program," *Proc. 3rd Int'l Joint Conf. on Artificial Intelligence*, pp. 73–85, 1973.

28. P. Bock, "The Emergence of Artificial Intelligence: Learning to Learn," *AI Magazine*, pp. 180–190, Fall 1985.

29. L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier Systems and Genetic Algorithms," Technical Report 8, Cognitive Science and Machine Intelligence Laboratory, University of Michigan, April 1987.

30. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth International Group, Belmont, CA, 1984.

31. B. G. Buchanan and T. M. Mitchell, "Model-Directed Learning of Production Rules," *Pattern-Directed Inference Systems*, eds. D. A. Waterman and F. Hayes-Roth, Academic Press, 1978.

32. J. G. Carbonell, "Experiential Learning in Analogical Problem Solving," *Proc. Nat'l Conf. on Artificial Intelligence*, pp. 168–171, 1982.

33. J. G. Carbonell and Y. Gil, "Learning by Experimentation," *Proc. 4th Int'l Machine Learning Workshop*, pp. 256–266, 1987.

34. G. A. Carpenter and S. Grossberg, "ART 2: Self-organization of Stable Category Recognition Codes for Analog Input Patterns," *Applied Optics*, vol. 26, pp. 4919–4930, December 1987.

35. A. C. Catania, *Learning*, Prentice Hall, Englewood Cliffs, NJ, 1979.

36. S. A. Chien, "Extending Explanation-Based Learning: Failure Driven Schema Refinement," Technical Report No. UILU-ENG-87-2203, University of Illinois, 1987.

37. T. C. Chow and J. A. Abraham, "Load Balancing in Distributed Systems," *Trans. on Software Engineering*, vol. SE-8, pp. 401–412, July 1982.

38. Y. C. Chow and W. Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *Trans. on Computers*, vol. C-28, pp. 334–361, May 1979.

39. J. Christensen and R. E. Korf, "A Unified Theory of Heuristic Evaluation Functions and Its Application to Learning," *Proc. 5th Nat'l Conf. on Artificial Intelligence AAAI-86*, pp. 148–152, 1986.

40. W. Clancey, "Heuristic Classification," in *Artificial Intelligence*, vol. 27, pp. 289–350, Amsterdam, 1985.

41. W. J. Clancey, "Classification Problem Solving," STAN-CS-84-1018, Stanford University, CA, July 1984.

42. G. C. Collins, "Plan Creation: Using Strategies as Blueprints," Ph.D. Thesis, Yale University, 1987.

43. M. E. Connell and P. E. Utgoff, "Learning to Control a Dynamic Physical System," *Proc. 6th Nat'l Conf. on Artificial Intelligence*, pp. 456–60, Seattle, WA, June 1987.

44. A. P. Danyluk, "The Use of Explanations for Similarity-Based Learning," *Proc. 10th Int'l Joint Conf. on Artificial Intelligence*, pp. 274–276, Milan, Italy, 1987.

45. T. R. Davies and S. J. Russell, "A Logical Approach to Reasoning by Analogy," *Proc. 10th Int'l Joint Conf. on Artificial Intelligence*, pp. 264–270, 1987.

46. R. Davis and R. B. Lenat, *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, New York, 1982.

47. T. Dean and K. Kanazawa, "Probabilistic Temporal Reasoning," *Proc. Nat'l Conf. on Artificial Intelligence AAAI-88*, pp. 524–528, 1988.

48. R. Dechter and J. Pearl, "Network-Based Heuristics for Constraint-Satisfaction Problems," *Artificial Intelligence*, vol. 34, pp. 1–38, 1988.

49. G. F. DeJong and R. J. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning*, vol. 1, pp. 145–176, 1986.

50. M. Derthick, "Counterfactual Reasoning with Direct Models," *Proc. Nat'l Conf. on Artificial Intelligence AAAI-87*, pp. 346–351, July 1987.

51. R. Desimone, "Learning Control Knowledge within an Explanation-Based Learning Framework," *Progress in Machine Learning*, eds. I. Bratko and N. Lavrac, pp. 107–119, Sigma Press, Cheshire, U.K., 1987.

52. T. G. Dietterich and B. G. Buchanan, "The Role of Critic in Learning Systems," Technical Report No. STAN-CS–81-891, Stanford University, CA, December 1981.

53. T. G. Dietterich and J. S. Bennett, "The Test Incorporation Theory of Problem Solving," *Proc. Workshop on Knowledge Compilation*, Oregon State University, Department of Computer Science, pp. 145–159, September 1986.

54. T. G. Dietterich, "Learning at the Knowledge Level," *Machine Learning*, vol. 1, pp. 287–316, Boston, 1986.

55. H. J. Einhorn, "Learning from Experience and Suboptimal Rules in Decision Making," *Cognitive Processes in Choice and Decision Behavior*, ed. T. Wallsten, Erlbaum, Hillsdale, NJ, 1980.
56. H. J. Einhorn and R. M. Hogarth, "Behavioral Decision Theory: Processes of Judgment and Choice," *Annual Review of Psychology*, vol. 32, pp. 53–88, 1981.
57. G. Ernst and A. Newell, *GPS: A Case Study in Generality and Problem Solving*, Academic Press, 1969.
58. G. W. Ernst and M. M. Goldstein, "Mechanical Discovery of Classes of Problem-Solving Strategies," *J. of the ACM*, vol. 29, pp. 1–23, January 1982.
59. S. E. Fahlman, G. E. Hinton, and T. J. Sejnowski, "Massively Parallel Architectures for AI: NETL, Thistle, and Boltzmann Machines," *Proc. Nat'l Conf. on Artificial Intelligence*, pp. 109–113, 1983.
60. B. Falkenhainer, "The Utility of Difference-Based Reasoning," *Proc. Nat'l Conf. on Artificial Intelligence AAAI-88*, Ann Arbor, Michigan, 1988.
61. E. A. Feigenbaum, "The Art of Artificial Intelligence–Themes and Case Studies of Knowledge Engineering," *Proc. 5th Int'l Joint Conf. on Artificial Intelligence*, pp. 1014–1029, Los Altos, CA, August 1977,
62. J. A. Feldman and Y. Yakimovsky, "Decision Theory and Artificial Intelligence: I. A Semantics-Based Region Analyzer," *Artificial Intelligence*, vol. 5, pp. 349–371, 1974.
63. J. A. Feldman and R. F. Sproull, "Decision Theory and Artificial Intelligence II: The Hungry Monkey," *Cognitive Science*, vol. 1, pp. 158–192, Norwood, NJ, 1977.
64. J. A. Feldman, "Dynamic Connections in Neural Networks," *Biological Cybernetics*, vol. 46, pp. 27–39, 1982.
65. R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Readings in Artificial Intelligence*, eds. B. L. Webber and N. Nilsson, pp. 231–249, Morgan Kaufmann, 1981.
66. N. S. Flann, "Improving Problem Solving Performance by Example Guided Reformulation of Knowledge," *Proc. 1st Workshop on Change of Representation*, 1988.
67. L. M. Fu and B. G. Buchanan, "Enhancing Performance of Expert Systems by Automated Discovery of Metarules," Technical Report No. HPP-84-38, Stanford University, CA, 1984.
68. S. Fujita, "Self-Organization in Distributed Operating System," *Abstracts 1st Annual INNS Meeting, (Neural Networks)*, vol. 1 (supplement 1), p. 93, 1988.
69. M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-completeness*, San Francisco, CA, 1979.
70. J. Gaschnig, "A Problem Similarity Approach to Devising Heuristics: First Results," *Readings in Artificial Intelligence*, eds. B. L. Webber and N. Nilsson. pp. 21–29, Morgan Kaufmann, Los Altos, CA, 1981.
71. M. R. Genesereth and N. J. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1987.
72. M. P. Georgeff, "Search Methods Using Heuristic Strategies," *Proc. Int'l Joint Conf. on Artificial Intelligence IJCAI-81*, pp. 563–568, 1981.
73. M. P. Georgeff, "Strategies in Heuristic Search," *Artificial Intelligence*, vol. 20, pp. 393–425, 1983.

74. M. P. Georgeff and A. L. Lansky, "Reactive Reasoning and Planning," *Proc. 6th Nat'l Conf. on Artificial Intelligence*, pp. 677–82, Seattle, WA, June 1987.
75. M. P. Georgeff, "Planning," *Annual Review of Computer Science*, pp. 359–400, Annual Reviews Inc., Palo Alto, CA, 1987.
76. A. Ginsberg, "Representation and Problem Solving: Theoretical Foundations," Technical Report No. CBM-TR-141, Rutgers University, October 1984.
77. R. M. Golden, "A Unified Framework for Connectionist Systems," *Biological Cybernetics*, vol. 58, no. 2, p. 109, 1988.
78. A. Golding, P. S. Rosenbloom, and J. E. Laird, "Learning General Search Control Rules from Outside Guidance," *Proc. 10th Int'l Joint Conf. on Artificial Intelligence*, pp. 334–337, Milan, Italy, 1987.
79. J. Gordon and E. H. Shortliffe, "A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space," *Artificial Intelligence*, vol. 26, pp. 323–357, 1985.
80. R. Greiner, "Learning by Understanding Analogies," Technical Report CSRI-188, University of Toronto, August 1986.
81. P. Haddawy, "A Variable Precision Logic Inference System Employing the Dempster-Shafer Uncertainty Calculus," Technical Report No. UIUCDCS-F-86-959, Department of Computer Science, University of Illinois, Urbana, IL, 1986.
82. S. Hanks, "Representing and Computing Temporally Scoped Beliefs," *Proc. Nat'l Conf. on Artificial Intelligence AAAI-88*, pp. 501–505, 1988.
83. B. Hayes-Roth and M. Hewett, "Learning Control Heuristics in BB1," Technical Report No. KSL 85-02, Knowledge Systems Lab., Stanford University, CA, January 1985.
84. B. Hayes-Roth, "A Blackboard Architecture for Control," *Artificial Intelligence*, vol. 26, pp. 251–321, July 1985.
85. F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, *Building Expert Systems*, Addison-Wesley, Reading, MA, 1983.
86. L. Hendricks, H. Oppewal, and C. Vlek, "Relative Importance of Scenario Information versus Frequency Information in the Judgment of Risk," Technical Report, University of Gronigen, The Netherlands, 1987.
87. G. E. Hinton, "Distributed Representations," Technical Report No. 84-157, Department of Computer Science, Carnegie-Mellon University, October 1984.
88. G. E. Hinton, T. J. Sejnowski, and D. H. Ackley, *Boltzmann Machine: Constraint Satisfaction Network that Learns*, Carnegie-Mellon University, Pittsburgh, PA, 1984.
89. G. E. Hinton, "Connectionist Learning Procedures," *Artificial Intelligence*, vol. 40, pp. 185–234, 1989.
90. J. H. Holland and J. S. Reitman, "Cognitive Systems Based on Adaptive Algorithms," *Pattern Directed Inference Systems*, eds. D. A. Waterman and F. Hayes-Roth, Academic Press, 1978.
91. J. H. Holland, "Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms," *Machine Learning II*, eds. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. pp. 593–623, Morgan Kaufmann, 1986.
92. J. J. Hopfield and D. W. Tank, "Computing with Neural Circuits: A Model," *Science*, pp. 625–633, August 1986.

93. E. J. Horvitz, "Reasoning about Beliefs and Actions under Computational Resource Constraints," *Proc. 3rd AAAI Workshop on Uncertainty in Artificial Intelligence*, Seattle, WA, July 1987.

94. K. Hwang, B. W. Wah, and F. A. Briggs, "Engineering Computer Network (ECN): A Hardwired Network of UNIX Computer Systems," *Proc. Nat'l Computer Conf.*, pp. 191–201, May 1981.

95. K. Hwang, W. J. Croft, G. H. Goble, B. W. Wah, F. A. Briggs, W. R. Simmons, and C. L. Coates, "A UNIX-based Local Computer Network with Load Balancing," *Computer*, vol. 15, pp. 55–66, April 1982. Also in *Tutorial: Computer Architecture*, eds. D. D. Gajski, V. M. Milutinovic, H. J. Siegel, and B. P. Furht, pp. 541–552, IEEE Computer Society, 1987.

96. K. B. Irani and J. Cheng, "Subgoal Ordering and Goal Augmentation for Heuristic Problem Solving," *Proc. 10th Int'l Joint Conf. on Artificial Intelligence*, pp. 1018–1024, Milan, Italy, 1987.

97. E. J. Johnson and J. W. Payne, "Effort and Accuracy in Choice," *Management Science*, vol. 31, April 1985.

98. B. Julish, F. Klix, R. Klein, W. Krause, and F. Kukla, "Some Experimental Results Regarding the Effect of Different Representations in Human Problem Solving," *Human and Artificial Intelligence*, ed. F. Klix, North-Holland, 1979.

99. D. Kahneman and A. Tversky, "Choices, Values, and Frames," *American Psychologist*, vol. 39, pp. 341–350, 1984.

100. H. A. Kautz, "The Logic of Persistence," *Proc. Nat'l Conf. on Artificial Intelligence AAAI-86*, p. 401, 1986.

101. D. S. Kay and J. B. Black, "The Evolution of Knowledge Representations with Increasing Expertise in Using Systems," *Proc. Cognitive Science Society Annual Meeting*, University of California, Irvine, 1985.

102. R. M. Keller, "A Survey of Research in Strategy Acquisition," Technical Report DCS-TR-115, Department of Computer Science, Rutgers University, May 1982.

103. D. Kibler and B. W. Porter, "Episodic Learning," *Proc. Nat'l Conf. on Artificial Intelligence AAAI-83*, pp. 191–196, 1983.

104. A. H. Klopf, "Drive-Reinforcement Learning: A Real-Time Mechanism for Classical Conditioning," *Proc. ICNN*, pp. II-441–II-445, 1987.

105. T. Kohonen, *Self-Organization and Associative Memory*, 2nd ed., Springer-Verlag, 1988.

106. M. M. Kokar and W. W. Zadrozny, "A Logical Model of Machine Learning," *Proc. 1st Workshop on Change of Representation*, 1988.

107. K. G. Konolige, "Experimental Robot Psychology," Technical Report No. 363, SRI International, Menlo Park, CA, November 1985.

108. R. E. Korf, *Learning to Solve Problems by Searching for Macro-Operators*, Pitman, Boston, 1985.

109. R. E. Korf, "Macro-Operators: A Weak Method for Learning," *Artificial Intelligence*, vol. 26, pp. 35–77, 1985.

110. R. E. Korf, "Heuristics as Invariants and its Application to Learning," *Machine Learning: A Guide to Current Research*, pp. 161–165, eds. T. M. Mitchell, J. G. Carbonell, and R. S. Michalski, Kluwer Academic, Boston, 1986.

111. B. Kosko, "Adaptive Bidirectional Associative Memories," *Applied Optics*, vol. 26, pp. 4947–4960, December 1987.

112. T. Kramer, "Automated Analysis of Operators on State Tables: A Technique for Intelligent Search," *Journal of Automated Reasoning*, vol. 2, pp. 127–151, 1986.

113. B. J. Kuipers, A. J. Moskowitz, and J. P. Kassirer, "Critical Decisions under Uncertainty," AI TR87-61, University of Texas at Austin, August 1987.

114. V. Kumar, "Branch-and-Bound Search," Technical Report AI TR85-11, University of Texas at Austin, August 1985.

115. J. E. Laird, P. S. Rosenbloom, and A. Newell, "Towards Chunking as a General Learning Mechanism," *Proc. Nat'l Conf. on Artificial Intelligence AAAI-84*, pp. 188–192, 1984.

116. J. E. Laird, P. S. Rosenbloom, and A. Newell, "Chunking in Soar: The Anatomy of a General Learning Mechanism," *Machine Learning*, vol. 1, pp. 11–46, 1986.

117. J. E. Laird, P. S. Rosenbloom, and A. Newell, "Soar: An Architecture for General Intelligence," *Artificial Intelligence*, vol. 33, pp. 1–64, 1987.

118. P. Langley and H. A. Simon, "The Central Role of Learning in Cognition," *Cognitive Skills and Their Acquisition*, ed. J. R. Anderson, Lawrence Erlbaum Associates, 1981.

119. P. Langley, "Learning Effective Search Heuristics," *Proc. 8th Int'l Joint Conf. on Artificial Intelligence*, pp. 419–421, Los Altos, CA, 1983.

120. P. Langley, "Learning to Search: From Weak Methods to Domain-Specific Heuristics," *Cognitive Science*, vol. 9, pp. 217–260, 1985.

121. D. Lawson and B. Williams, "A Neural Network Implementation of a Page-Swapping Algorithm," *Abstracts of the 1st Annual INNS Meeting (Neural Networks)*, vol. 1 (supplement 1), p. 451, 1988.

122. M. Lebowitz, "Integrated Learning: Controlling Explanation," *Cognitive Science*, pp. 219–240, 1986.

123. K. F. Lee and S. Mahajan, "A Pattern Classification Approach to Evaluation Function Learning," *Artificial Intelligence*, vol. 36, pp. 1–25, 1988.

124. D. B. Lenat, "HEURETICS: Theoretical and Experimental Study of Heuristic Rules," *Proc. Nat'l Conf. on Artificial Intelligence AAAI-82*, pp. 159–163, 1982.

125. D. B. Lenat, "The Role of Heuristics in Learning by Discovery: Three Case Studies," *Machine Learning: An Artificial Intelligence Approach*, eds. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, pp. 243–306, Morgan Kaufmann, Los Altos, CA, 1983.

126. D. B. Lenat, "Theory Formation by Heuristic Search; The Nature of Heuristics II: Background and Examples," *Artificial Intelligence*, vol. 21, pp. 31–59, 1983.

127. D. B. Lenat and J. S. Brown, "Why AM and EURISKO Appear to Work," *Artificial Intelligence*, vol. 23, pp. 269–294, 1984.

128. D. B. Lenat and E. A. Feigenbaum, "On the Thresholds of Knowledge," *Proc. 10th Int'l Joint Conf. on Artificial Intelligence*, pp. 1173–1182, 1987.

129. C. Lewis, "Composition of Productions," *Production System Models of Learning and Development*, ed. Klahr et al. , MIT Press, 1987.

130. M. W. Lewis and J. R. Anderson, "Discrimination of Operator Schemata in Problem Solving: Learning from Examples," *Cognitive Psychology*, vol. 17, pp. 26–65, 1985.

131. M. R. Lowry, "The Logic of Problem Reformulation," *Proc. Workshop on Knowledge Compilation*, ed. T. G. Dietterich, 1986.

132. F. Maffioli, "The Complexity of Combinatorial Optimization Algorithms and the Challenge of Heuristics," *Combinatorial Optimization*, pp. 107–128, 1979.

133. Z. Manna and R. Waldinger, "A Theory of Plans," *Reasoning about Actions and Plans*, eds. M. P. Georgeff and A. L. Lansky, pp. 11–45, 1987.

134. J. L. McClelland, "The Programmable Blackboard Model of Reading," *Parallel Distributed Processing: Psychological and Biological Models*, eds. J. L. McClelland and D. E. Rumelhart, pp. 122–169, MIT Press, 1986.

135. T. L. McCluskey, "Combining Weak Learning Heuristics in General Problem Solvers," *Proc. 10th Int'l Joint Conf. on Artificial Intelligence*, pp. 331-333, Milan, Italy, 1987.

136. L. Mero, "A Heuristic Search Algorithm with Modifiable Estimate," *Artificial Intelligence*, vol. 23, pp. 13–27, 1984.

137. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine Learning: An Artificial Intelligence Approach*, 1983.

138. R. S. Michalski, "A Theory and Methodology of Inductive Learning," *Machine Learning*, ed. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Tioga, 1983.

139. D. Michie and R. Chambers, "BOXES: An Experiment in Adaptive Control," *Machine Intelligence 2*, pp. 137–152, eds. E. Dale and D. Michie, Oliver and Boyd, Edinburgh, Scotland.

140. M. Minsky, "Steps toward Artificial Intelligence," *Computers and Thought*, eds. E. A. Feigenbaum and J. Feldman, McGraw-Hill, New York, 1963.

141. S. Minton, J. G. Carbonell, C. A. Knoblock, D. Kuokka, and H. Nordin, "Improving the Effectiveness of Explanation Based Learning," *Proc. Workshop on Knowledge Compilation*, pp. 77–87, 1986.

142. S. Minton and J. G. Carbonell, "Strategies for Learning Search Control Rules: An Explanation-Based Approach," *Proc. 10th Int'l Joint Conf. on Artificial Intelligence*, pp. 334–337, Milan, Italy, August 1987.

143. T. M. Mitchell, "The Need for Biases in Learning Generalizations," CBM-TR-117, Rutgers University, Computer Science Department, 1980.

144. T. M. Mitchell, P. E. Utgoff, B. Nudel, and R. Benerji, "Learning Problem-Solving Heuristics through Practice," *Proc. 7th Int'l Joint Conf. on Artificial Intelligence*, pp. 127–134, Los Altos, CA, 1981.

145. T. M. Mitchell, "Learning and Problem Solving," *Proc. 8th Int'l Joint Conf. on Artificial Intelligence*, pp. 1139-1151, Los Altos, CA, August 1983.

146. T. M. Mitchell, P. E. Utgoff, and R. B. Banerji, "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics," *Machine Learning*, eds. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Tioga, 1983.

147. T. M. Mitchell, "Toward Combining Empirical and Analytical Methods for Inferring," *Artificial and Human Intelligence*, pp. 81–103, eds. R. B. Banerji and T. Elithorn, Elsevier, 1984.

148. R. J. Mooney, "A General Explanation-Based Learning Mechanism and Its Application to Narrative Understanding," Technical Report UILU-ENG-87-2269, University of Illinois, Urbana-Champaign, December 1987.

149. L. Morgenstern and L. A. Stein, "Why Things Go Wrong: A Formal Theory of Causal Reasoning," *Proc. Nat'l Conf. on Artificial Intelligence AAAI-88*, pp. 518–523, 1988.

150. D. J. Mostow, "Machine Transformation of Advice into a Heuristic Search Procedure," *Machine Learning: An Artificial Intelligence Approach*, eds. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, pp. 367–404, Morgan Kaufmann, Los Altos, CA, 1983.

151. B. A. Nadel, "Representation Selection for Constraint Satisfaction Problems: A Case Study Using n-Queens," DCS-TR-208, Laboratory for Computer Science Research, Rutgers University, NJ, 1987.

152. D. M. Neves and J. R. Anderson, "Knowledge Compilation: Mechanisms for the Automatization of Cognitive Skills," *Cognitive Skills and Their Acquisition*, ed. J. R. Anderson, Lawrence Erlbaum Associates, 1981.

153. D. M. Neves, "Learning Procedures from Examples and by Doing," *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 624–630, 1983.

154. A. Newell and H. A. Simon, *Human Problem Solving*, Prentice Hall, Englewood Cliffs, NJ, 1972.

155. A. Newell, "The Knowledge Level," *Artificial Intelligence*, vol. 18, pp. 87–127, 1982.

156. L. M. Ni and K. Hwang, "Optimal Load Balancing Strategies for a Multiple Processor System," *Proc. 10th Int'l Conf. on Parallel Processing*, pp. 352–357, August 1981.

157. L. M. Ni, C.-M. Xu, and T. B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," *Trans. on Software Engineering*, vol. SE-11, pp. 1153–1161, October 1985.

158. H. P. Nii, "Blackboard Systems, Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective," *AI Magazine*, pp. 82–106, August 1986.

159. N. J. Nilsson, "Triangle Tables: A Proposal for a Robot Programming Language," Technical Note 347, SRI International, Menlo Park, CA, February 1985.

160. S. Ohlsson, "A Constrained Mechanism for Procedural Learning," *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 426–428, Los Altos, CA, 1983.

161. P. S. Ow, S. F. Smith, and A. Thiriez, "Reactive Plan Revision," *Proc. 10th Nat'l Conf. on Artificial Intelligence AAAI-88*, vol. 1, pp. 77–82, Saint Paul, MN, 1988.

162. V. M. Ozernoi and M. G. Gaft, "Methods for the Best Solutions Search in Multi-objective Decision Problems," *Proc. Int'l Joint Conf. on Artificial Intelligence IJCAI-75*, pp. 357–361, Moscow, USSR, 1975.

163. J. C. Pasquale, "Intelligent Decentralized Control in Large Distributed Computer Systems," Ph.D. Thesis, University of California, Berkeley, April 1988.

164. J. W. Payne, J. R. Bettman, E. J. Johnson, and E. Coupey, "Selection of Heuristics for Choice: An Effort and Accuracy Perspective," *Illinois Interdisciplinary Workshop on Decision Making*, 1988.

165. M. J. Pazzani, "Explanation and Generalization Based Memory," *Proc. Annual Meeting of the Cognitive Science Society*, pp. 323–328, Irvine, CA, 1985.

166. J. Pearl, "Some Recent Results in Heuristic Search Theory," *Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, pp. 1–13, January 1984.

167. J. Pearl, *Heuristics—Intelligent Search Strategies for Computer Problem Solving*, Addison Wesley, Reading, MA, 1984.

168. J. Pearl, "Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning," *Proc. 7th Annual Conf. of Cognitive Science Society*, pp. 329–334, August 1985.

169. G. D. Plotkin, "A Note on Inductive Generalization," *Machine Intelligence*, ed. Meltzer and Michie, pp. 153–163, Edinburgh University Press, Edinburgh, Scotland, 1970.

170. I. Pohl, "The Avoidance of (Relative) Catastrophe, Heuristic Competence, Genuine Dynamic Weighting, and Computational Issues in Heuristic Problem Solving," *Proc. of IJCAI–73*, pp. 12–17, Stanford, CA, 1973.

171. M. E. Pollack, D. J. Israel, and M. E. Bratman, "Toward an Architecture for Resource-Bounded Agents," CSLI-87-104, Center for the Study of Language and Information, Menlo Park, CA, August 1987.

172. B. Porter and D. Kibler, "Experimental Goal Regression: A Technique for Learning Heuristics," Technical Report AITR 86-20, Department of Computer Science, University of Texas, Austin, 1986.

173. B. W. Porter and D. F. Kibler, "Learning Operator Transformations," *Proc. Nat'l Conf. on Artificial Intelligence AAAI–84*, pp. 278–282, 1984.

174. B. W. Porter and R. E. Bareiss, "PROTOS: An Experiment in Knowledge Acquisition for Heuristic Classification Tasks," Technical Report AI TR-86-35, A.I. Lab., University of Texas at Austin, September 1986.

175. J. R. Quinlan, "Predicting the Length of Solutions to Problems," *Proc. Int'l Joint Conf. on Artificial Intelligence IJCAI-75*, pp. 363–369, 1975.

176. S. Rajamoney, G. DeJong, and B. Faltings, "Towards a Model of Conceptual Knowledge Acquisition through Directed Experimentation," Working Paper 68, Artificial Intelligence Research Group, Coordinated Science Lab., University of Illinois, 1985.

177. S. A. Rajamoney and G. F. DeJong, "Active Ambiguity Reduction: An Experiment Design Approach to Tractable Qualitative Reasoning," Technical Report No. UILU-ENG-87- 2225, University of Illinois, April 1987.

178. S. A. Rajamoney, "Exemplar-Based Theory Rejection: An Approach to the Experience Consistency Problem," *Machine Learning*, pp. 284–289, Boston, Kluwer, 1989.

179. L. Rendell, "Conceptual Knowledge Acquisition in Search," *Computational Models of Learning*, ed. L. Bolc, Springer Verlag, 1987.

180. L. A. Rendell, "A New Basis for State-Space Learning Systems and a Successful Implementation," *Artificial Intelligence*, vol. 20, pp. 369-392, 1983.

181. L. A. Rendell, "Substantial Constructive Induction Using Layered Information Compression: Tractable Feature Formation in Search," *Proc. Int'l Joint Conf. on Artificial Intelligence IJCAI-85*, p. 65, 1985.

182. L. A. Rendell, "Genetic Plans and the Probabilistic Reasoning System: Synthesis and Results," UIUCDCS-R-85-1217, Department of Computer Science, University of Illinois, 1985.

183. L. A. Rendell, "A Framework for Induction and a Study of Selective Induction," *Machine Learning*, vol. 1, June 1986.

184. L. A. Rendell, "Representations and Models for Concept Learning," UIUCDCS-R-87-1324, Department of Computer Science, University of Illinois, 1987.

185. L. A. Rendell, R. Seshu, and D. Tcheng, "Layered Concept Learning and Dynamically Variable Bias Management," *Proc. Int'l Joint Conf. on Artificial Intelligence IJCAI-87*, Milan, Italy, 1987.

186. L. A. Rendell, "Learning Hard Concepts," *European Workshop in Learning* (*EWSL-88*), 1988.

187. P. J. Riddle, "An Overview of Problem Reduction: A Shift of Problem Representation," *Proc. Workshop on Knowledge Compilation*, ed. T. G. Dietterich, 1986.

188. P. J. Riddle, "An Approach for Learning Problem Reduction Schemas and Iterative Macro-Operators," *Proc. 1st Workshop on Change of Representation*, 1988.

189. M. H. Romanycia and F. J. Pelletier, "What Is a Heuristic?," *J. Computational Intelligence*, vol. 1, pp. 47–58, Toronto, 1985.

190. F. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, New York, 1962.

191. P. Rosenbloom and A. Newell, "Learning by Chunking: A Production System Model of Practice," *Production System Models of Learning and Development*, ed. Klahr et al., MIT Press, 1987.

192. P. S. Rosenbloom, J. E. Laird, J. McDermott, A. Newell, and E. Orciuch, "R1-Soar: An Experiment in Knowledge-Intensive Programming in a Problem Solving Architecture," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-7, pp. 561–569, 1985.

193. P. S. Rosenbloom and J. E. Laird, "Mapping Explanation-Based Generalization onto SOAR," STAN-CS-86-1111 (also KSL-86-46), Department of Computer Science, Stanford University, 1986.

194. R. Rosenfeld and D. S. Touretzky, "Four Capacity Models for Coarse-Coded Symbol Memories," Technical Report No. CMU-CS-87-182, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, December 1987.

195. J. Rosenschein and V. Singh, "The Utility of Metalevel Effort," HPP-83-20, Stanford University, CA, March 1983.

196. D. D. Rumelhart, G. Hinton, and J. L. McClelland, "A General Framework for Parallel Distributed Processing," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, eds. D. E. Rumelhart, J. L McClelland, and the PDP Research Group, MIT Press, Cambridge, MA, 1986.

197. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*, Institute for Cognitive Science Report 8506, UCSD, September 1985.

198. D. E. Rumelhart, P. Smolensky, J. L. McClelland, and G. Hinton, "Schemata and Sequential Thought in PDP Models," *Parallel Distributed Processing: Psychological and Biological Models*, MIT Press, 1986.

199. J. E. Russo and B. A. Dosher, "Strategies for Multiattribute Binary Choice," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 9, pp. 676–696, 1983.

200. E. D. Sacerdoti, *A Structure for Plans and Behavior*, Elsevier, New York, 1977.

201. E. D. Sacerdoti, "Problem Solving Tactics," *Proc. IJCAI-79*, pp. 1077–1085, 1979.

202. A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM J. Research and Development* , vol. 3, pp. 210–229, 1959.

203. A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers II—Recent Progress," *J. of Research and Development*, vol. 11, pp. 601–617, 1967.

204. R. C. Schank, G. C. Collins, and L. Hunter, "Transcending Inductive Category Formation in Learning," *Behavioral and Brain Sciences*, vol. 9, pp. 639–686, 1986.

205. M. J. Schoppers, "Representation and Automatic Synthesis of Reaction Plans," forthcoming Ph.D. thesis, Department of Computer Science, University of Illinois, Urbana, 1988.

206. B. Silver, *Meta-Level Inference: Representing and Learning Control Information in Artificial Intelligence, Studies in CS and AI*, New York: North-Holland, 1986.

207. H. A. Simon, "The Architecture of Complexity," *The Sciences of the Artificial*, pp. 193–230, MIT Press, Cambridge, MA, 1969.

208. H. A. Simon, "Economic Rationality: Adaptive Artifice," *The Sciences of the Artificial*, p. 46, MIT Press, Cambridge, MA, 1969.

209. H. A. Simon, "Artificial Intelligence Systems that Understand," *Proc. 5th Int'l Joint Conf. on Artificial Intelligence*, vol. 2, pp. 1059–1073, Cambridge, MA, 1977.

210. H. A. Simon, "Information-Processing Theory of Human Problem Solving," *Handbook of Learning and Cognitive Processes*, Erlbaum, 1978.

211. H. A. Simon, "What the Knower Knows: Alternative Strategies for Problem Solving Tasks," *Human and Artificial Intelligence*, ed. F. Klix, North-Holland, 1979.

212. H. A. Simon, *The Science of the Artificial*, 2nd edition, MIT Press, Cambridge, MA, 1981.

213. H. A. Simon, "Why Should Machines Learn?" *Machine Learning: An Artificial Intelligence Approach*, eds. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, pp. 25–37, Morgan Kaufmann, Los Altos, CA, 1983.

214. J. R. Slagle, "A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus," *Computers and Thought*, eds., E. A. Feigenbaum and J. Feldman, McGraw-Hill, New York, 1963.

215. D. Sleeman, P. Langley, and T. M. Mitchell, "Learning from Solution Paths: An Approach to the Credit Assignment Problem," *AI Magazine*, vol. 3, pp. 48–52, 1982.

216. R. G. Smith, T. M. Mitchell, R. A. Chestek, and B. G. Buchanan, "A Model for Learning Systems," *Proc. 5th Int'l Joint Conf. on Artificial Intelligence*, pp. 338–343, Los Altos, CA, August 1977.

217. S. F. Smith, "Flexible Learning of Problem Solving Heuristics through Adaptive Search," *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 422–425, 1983.

218. P. Smolensky, "Information Processing in Dynamical Systems: Foundations of Harmony Theory," *Parallel Distributed Processing: Foundations*, pp. 194–281, MIT Press, 1986.

219. N. S. Sridharan and J. L. Bresina, "Exploration of Problem Reformulation and Strategy Acquisition—A Proposal," Laboratory for Computer Science Research, Rutgers University, March 1984.

220. S. Star, "Theory-Based Inductive Learning: An Integration of Symbolic and Quantitative Methods," *Uncertainty in Artificial Intelligence Workshop*, pp. 237–248, Seattle, WA, 1987.

221. M. Stefik, "Planning and Meta-Planning (MOLGEN: Part 2)," *Readings in Artificial Intelligence*, pp. 272–286, eds. B. L. Webber and N. Nilsson, Morgan Kaufmann, 1981.

222. D. Steier, "CYPRESS-Soar: A Case Study in Search and Learning in Algorithm Design," *Proc. 10th Int'l Joint Conf. on Artificial Intelligence IJCAI-87*, pp. 327–330, Milan, Italy, 1987.

223. D. Subramanian and M. R. Genesereth, "Reformulation," *Proc. Workshop on Knowledge Compilation*, ed., T. G. Dietterich, 1986.

224. G. J. Sussman, "A Computational Model of Skill Acquisition," Ph.D. thesis, Artificial Intelligence Lab., MIT, 1973.

225. R. S. Sutton, "Temporal Credit Assignment in Reinforcement Learning," Ph.D. thesis, University of Massachusetts, Amherst, February 1984.

226. R. S. Sutton and A. G. Barto, "Toward a Modern Theory of Adaptive Networks: Expectation and Prediction," *Psychological Review*, vol. 88, pp. 135–170, 1984.

227. R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

228. C. Swart and D. Richards, "On the Inference of Strategies," TR 86-20-3, Oregon State University Computer Science Department, 1986.

229. D. S. Touretzky and G. E. Hinton, "A Distributed Connectionist Production System," CMU-CS-86-172, Department of Computer Science, Carnegie-Mellon University, 1986.

230. D. S. Touretzky, "Beyond Associative Memory: Connectionists Must Search for Other Cognitive Primitives," *Proc. AAAI Spring Symp. Series: Parallel Models of Intelligence: How Can Slow Components Think So Fast?*, Stanford, CA, 1988.

231. A. Tversky and D. Kahneman, "Judgment under Uncertainty: Heuristics and Biases," *Science*, vol. 185, pp. 1124–1131, 1974.

232. P. E. Utgoff and T. M. Mitchell, "Acquisition of Appropriate Bias for Inductive Concept Learning," *Proc. Nat'l Conf. on Artificial Intelligence*, pp. 414–417, 1982.

233. P. E. Utgoff, "Adjusting Bias in Concept Learning," *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 447–449, Los Altos, CA, 1983.

234. P.E. Utgoff and S. Saxena, "Learning a Preference Predicate," *Proc. Int'l Machine Learning Workshop*, pp. 115–121, Irvine, CA, June 1987.

235. B. W. Wah and P. Mehra, "Learning Parallel Search in Load Balancing," *Workshop on Parallel Algorithms for Machine Intelligence and Pattern Recognition*, Minneapolis, MN, August 21, 1988.

236. B. Walker, G. Popek, R. English, C. Kline, and G. Thiel, "The LOCUS Distributed Operating System," *Proc. 9th Symp. on Operating System Principles*, pp. 49–70, 1983.

237. Y. T. Wang and J. T. Morris, "Load Sharing in Distributed Systems," *Trans. on Computers*, vol. C-34, pp. 204–217, March 1985.

238. S. Watanabe, *Pattern Recognition: Human and Mechanical*, Wiley Interscience, 1985.

239. S. Watanabe, "Inductive Ambiguity and the Limits of Artificial Intelligence," *Computational Intelligence*, vol. 3, pp. 304–309, 1987.

240. D. A. Waterman, "Generalization Learning Techniques for Automating the Learning of Heuristics," *Artificial Intelligence*, vol. 1, pp. 121–170, 1970.

241. P. J. Werbos, "Learning How the World Works: Specifications for Predictive Networks in Robots and Brains," *Proc. IEEE Int'l Conf. on Systems, Man, and Cybernetics*, vol. 1, pp. 302–310, October 1987.

242. P. J. Werbos, "Backpropagation: Past and Future," *Proc. Int'l Conf. on Neural Networks ICNN-88*, San Diego, CA, July 1988.

243. B. L. Whitehall, "Incremental Learning with INDUCE/NE," Working Paper 84, AI Group, Coordinated Science Lab, University of Illinois, 1986.

244. B. L. Whitehall, "Substructure Discovery in Executed Action Sequences," Technical Report UILU-ENG-87-2256, University of Illinois, College of Engineering, September 1987.

245. C. C. White III and S. Dozono, "A Generalized Model of Sequential Decision-Making under Risk," *European Journal of Operational Research*, vol. 18, pp. 19–26, 1984.

246. D. C. Wilkins, "Knowledge Base Refinement by Monitoring Abstract Control Knowledge," *Knowledge Acquisition for Knowledge Based Systems*, ed. Boose et al., Academic Press, New York, 1987.

247. D. C. Wilkins, "Knowledge Base Refinement Using Apprenticeship Learning Techniques," *Proc. Nat'l Conf. on Artificial Intelligence AAAI-88*, 1988.

248. R. J. Williams, "On the Use of Backpropagation in Associative Reinforcement Learning," *Proc. 2nd Int'l Conf. on Neural Networks ICNN-88*, vol. 1, pp. 263–270, San Diego, CA, July 1988.

249. S. W. Wilson, "Hierarchical Credit Allocation in Classifier Systems," *Genetic Algorithms and Simulated Annealing*, ed. L. Davis, Pitman, London, 1987.

250. P. H. Winston, "Learning Structural Descriptions from Examples," *The Psychology of Computer Vision*, ed. P. H. Winston, McGraw-Hill, New York, 1975.

251. P. H. Winston, T. O. Binford, B. Katz, and M. R. Lowry, "Learning Physical Descriptions from Functional Definitions, Examples, and Precedents," *Proc. AAAI-83*, pp. 433–439, 1983.

252. P. H. Winston, *Artificial Intelligence*, Addison-Wesley, Reading, MA, 1984.

253. C. F. Yu, *Efficient Combinatorial Search Algorithms*, Ph.D. thesis, School of Electrical Engineering, Purdue University, West Lafayette, IN, December 1986.

254. C. F. Yu and B. W. Wah, "Learning Dominance Relations in Combinatorial Search Problems," *Trans. on Software Engineering*, vol. SE-14, no. 8, August 1988.

255. B. Zhang and L. Zhang, "The Statistical Inference Method in Heuristic Search Techniques," *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 757–759, 1983.