International Journal on Artificial Intelligence Tools, Vol. 7, No. 4 (1998) 487–550
© World Scientific Publishing Company

STRATEGY LEARNING: A SURVEY OF PROBLEMS, METHODS, AND ARCHITECTURES

PANKAJ MEHRA
Tandem Labs.
10555 Ridgeview Court
Cupertino, CA 95014-0789, USA
mehra_pankaj@esp.tandem.com

BENJAMIN W. WAH
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, IL 61801, USA
b-wah@uiuc.edu

Received 30 June 1996
Accepted 5 August 1998

Problem solvers employ strategies in searching for solutions to given problem instances. Strategies have traditionally been designed by experts using prior knowledge and refined manually using trial and error. Recent attempts to automate these processes have produced *strategy-learning systems*. This paper shows how various issues in strategy learning are affected by the nature of performance tasks, problem solvers, and learning environments. Surveyed learning systems are grouped by the commonality of their approaches into four general architectures.

Keywords: Artificial intelligence, credit assignment, delayed feedback, heuristics, ill-defined objectives, learning architectures, machine learning, sequential problems, strategies.

1 Introduction

Much recent work in machine learning has targeted *sequential problems* in decision, control, and optimization. Sequential problems abound in robotics, navigation, tracking, and dynamic scheduling. They require the problem solver to make multiple choices or initiate multiple actions, one after another, in order to reach and/or maintain a "desired state." Such a sequence of choices or actions constitutes a solution; the knowledge used for generating it is a *strategy*. Problem solvers employ strategies to systematically generate solutions to given problem instances. Our focus in this paper is on the automated learning of such strategies.

Strategies are useful when there are choices, and guidance is needed in exploring or searching various courses of action. When a single course of action is clear at the outset, the problem can be solved algorithmically. Otherwise, systematic exploration or *search* is performed on the space of alternatives. Search without guidance is often prohibitively expensive and, therefore, intractable. Search with guidance is called *heuristic search* [1]. *Heuristics*, in this context, are modular pieces of readily applicable knowledge — "rules of thumb" [2, 3] — which allow the decision maker to select, prefer, or rule out some alternatives at each decision point. A strategy, on the other hand, is a "general plan of action" that applies to multiple decision points [4]. Frequently, strategies are broken down into modular heuristic rules that are applied dynamically; this is especially true for applications with time-varying inputs. Perhaps that is why some researchers do not distinguish between heuristics and strategies [2].

We will not distinguish between learning to improve solution quality and learning to improve problem-solving speed because both notions can be captured equally well by a properly designed objective function and reward/penalty scheme.

1.1 State of the Art

Traditionally, strategies have been designed by experts using prior knowledge, and refined manually using trial and error. Recent attempts to automate these processes have produced *strategy-learning systems* that run the gamut of applications from dynamic load balancing to symbolic reasoning and combinatorial optimization. Despite its volume and diversity, the literature on strategy-learning systems lacks a systematic characterization of the relationship between applications, algorithms, and architectures. Prior surveys [5, 6, 7, 8] have focused on well-defined symbolic learning problems characterized by simple feedback schemes and knowledge-rich learning environments.

Time is an important parameter of sequential behavior. Systems that learn strategies for sequential problems must, therefore, tackle a number of temporal problems. These generally involve predicting future states of the environment. The states may evolve under the influence of either the problem solver's decisions (*causal dynamics*), or the passage of time (*natural dynamics*), or both; difficult prediction problems result in the last case. Learning systems developed by artificial intelligence (AI) researchers have had limited success in coping with natural dynamics.

When learning strategies by trial and error, a learning system may experiment with multiple strategies. The measured performance of tested strategies provides *feedback* for guiding both the selection of strategies for future tests and the modification of incumbent strategies toward improved performance. The translation of feedback into strategy modifications is called *credit assignment* [9]. Successive occurrences of performance feedback often span an entire sequence of decisions. The credit/blame implied by the feedback signal must be distributed temporally (among the decisions) as well as structurally (among the various factors governing each decision). Sutton [10] was the first to make this distinction between temporal and structural credit assignment (SCA). Credit assignment requires extensive knowledge of the problem domain. The temporal credit-assignment (TCA) prob-

Table 1: Components of strategy-learning problems.

Component	Description
Performance Task	Defines a class of problems whose instances are to be solved
Problem Solver	Solves instances of performance task using strategies to decide what operation(s) to perform at each decision point.
Learning Environment	Reacts to problem solver's actions by providing feedback; may also provide prior knowledge of problem domain.

lem (like the projection problems described above) is complicated by time-varying states. Existing systems solve this problem using either ad hoc schemes [11, 12], user-supplied domain knowledge [13], or *Markov property* (future states may depend on the current state but not on any of the past states) [14]. There is no general yet rational solution to the temporal credit-assignment problem.

Since closed-form objective functions may not be available for certain problems, strategy-learning systems sometimes optimize *measured* performance. The transformation of such (ill-posed) problems into well-defined ones, which the problem solver can solve, is often achieved by empirically modeling the true objective function from data. Several existing systems [15, 16, 17, 18] learn evaluation functions that associate a scalar performance metric with every state of the problem solver. When the objective-function value depends on a state sequence rather than on a single state, one needs to learn scalar-valued functions of time series. This situation arises only when the state-space representation used by the problem solver is non-Markovian, *i.e.*, the quality of future states (solutions) depends not only on the current state but also on the path from the initial state to the current state. The problem of learning evaluation functions for non-Markovian representations has not been addressed in the learning literature.

The challenging problems in strategy learning are, therefore, the development of techniques to cope with time-varying parameters (natural dynamics), non-Markovian representations, and ill-defined objective functions.

1.2 Contributions of this Paper

We introduce attributes for classifying the problems, methods, and architectures of strategy-learning systems. We use the term ***strategy-learning problem*** to denote a triple $(\mathcal{P}, \mathcal{PS}, \mathcal{E})$, comprising a performance task $(\mathcal{P})$, a problem solver $(\mathcal{PS})$, and a learning environment $(\mathcal{E})$ (Table 1). Various aspects of $\mathcal{P}$, $\mathcal{PS}$, and $\mathcal{E}$ dictate the form and complexity of learning algorithms and system architectures. Studying these aspects is important for both characterizing the limitations of existing systems and finding an appropriate learning system for a given application.

Figure 1 shows a coarse classification of strategy-learning problems. Broadly, learning problems can be classified as either well-posed or ill-posed on the basis of the objective functions of their performance tasks. Ill-posed learning problems have performance tasks with *ill-defined objective functions*; that is, the objective

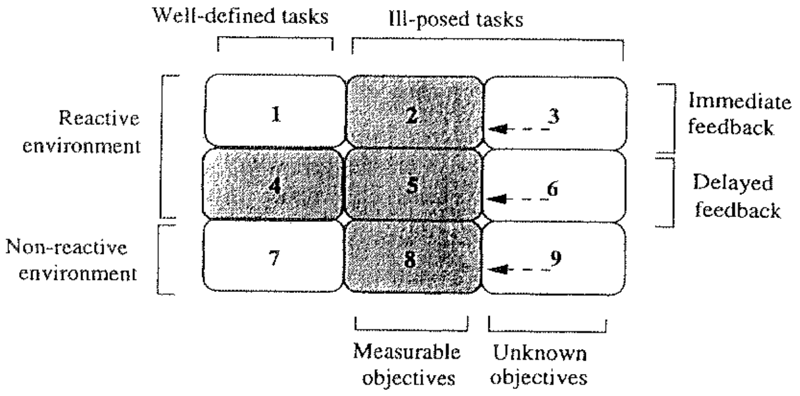


Figure 1: Classification of strategy-learning problems. (Shaded areas in this figure indicate classes of strategy-learning problems addressed in this paper.)

functions are not specified as closed-form functions of the problem solver's inputs. Ill-defined objective functions can be further classified as either measurable or unknown. Orthogonally, one may classify learning problems on the basis of the feedback structure of their learning environments. Here, learning environments can be classified as either reactive (those that produce feedback) or non-reactive (those that don't). Further, feedback may be immediate (which occurs regularly after each decision) or delayed (which occurs intermittently).

Our focus in this paper is on the classes of strategy-learning problems shown shaded in Figure 1. These problems are characterized by slow reactive learning environments and performance tasks having ill-defined but measurable objective functions. The unshaded boxes in the third column of Figure 1 correspond to problems with one or more unknown objective functions in their performance tasks. For such problems, it is not clear at the outset what function to optimize. A common technique to solve a problem in this class is to transform it into another with either a well-defined or an ill-defined but measurable objective function. This transformation is indicated by the dashed arrows in Figure 1. Another common technique is to add constraints to the problem and to solve for a feasible solution of the constrained problem. Finally, the unshaded boxes in column 1 represent well-posed problems with immediate feedback as well as those without feedback. Such problems have been studied extensively and can be learned by existing learning techniques in AI [19].

In Section 2, we analyze the structure of strategy-learning problems and identify the key issues in strategy learning. Section 3 describes techniques for addressing these issues, and Section 4 compares four architectures of strategy-learning systems.

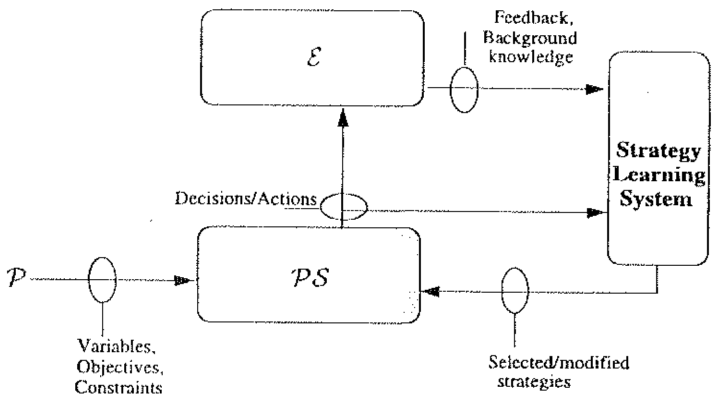

$\mathcal{E}$
Feedback, Background knowledge
Strategy Learning System
Decisions/Actions
$\mathcal{P}$
$\mathcal{PS}$
Variables, Objectives, Constraints
Selected/modified strategies

Figure 2: Typical information flow in a strategy-learning system.

2 The Anatomy of Strategy-Learning Problems

We view strategy-learning problems as triples $(\mathcal{P}, \mathcal{PS}, \mathcal{E})$, comprising a performance task $(\mathcal{P})$, a problem solver $(\mathcal{PS})$, and a learning environment $(\mathcal{E})$. It is important to distinguish between a performance task and its instances: each *instance* corresponds to some initial assignment of values to the problem variables (inputs) of $\mathcal{P}$. In this sense, $\mathcal{P}$ defines a *class* of problems whose instances are solved by $\mathcal{PS}$. When presented an instance of $\mathcal{P}$, $\mathcal{PS}$ instantiates the decision variables (outputs) of $\mathcal{P}$, thereby causing action via application of problem-solving operators. Operators transform both the external state of $\mathcal{E}$ and the internal state of $\mathcal{PS}$. $\mathcal{PS}$ reacts to the new external state by either applying another operator or stopping; it uses strategies in deciding what operator to apply next. Sometimes, $\mathcal{E}$ responds to operator application with *feedback* indicating the quality of current and recent external states. The role of a strategy-learning system (Figure 2) is to use the feedback received from $\mathcal{E}$ in order to improve the strategies used by $\mathcal{PS}$ in such a fashion that i) future actions will produce more favorable feedback, and ii) optimal or near-optimal stopping states (if any) will be reached quickly.

Besides modifying existing strategies, a learning system can improve the problem solver in several other ways. For example, it can i) learn a model of the objective function; or ii) learn to predict future states; or iii) learn to predict the state changes caused by a proposed action; or iv) learn to predict future feedback. The first of these is critical for ill-posed tasks; the second, for coping with natural dynamics; the third, for coping with causal dynamics; and the fourth, for solving difficult TCA problems. Whether or not a strategy-learning problem will be ill-posed depends on the objective function of $\mathcal{P}$; similarly, whether or not it will have natural dynamics depends on the variables of $\mathcal{P}$. A strategy-learning system will need learning to predict future states only if a causal model of $\mathcal{PS}$ is not provided as background knowledge by $\mathcal{E}$. Finally, prediction of future feedback is necessary only when feedback is delayed. Thus, the components of a strategy-learning problem strongly determine the specific issues that a learning system must address.

Before we begin to dissect and classify strategy-learning problems, we introduce a few problems drawn from diverse domains which will be used as running examples throughout this paper.

Example 1. *Learning Strategies for Load Balancing.* Consider a set of workstations connected by a local area network. Users at each workstation (or site) issue commands interactively; each command spawns off one or more jobs. The objective is to finish these jobs as quickly as possible. Through a technique called load balancing, the time taken to complete a job can be reduced by 'migrating' some jobs from heavily loaded sites to lightly loaded ones. A job's completion time depends not only upon the raw speed of the site at which it is executed but also upon the 'background load' at that site due to competing jobs. The performance task $\mathcal{P}$ is to decide whether to run a job locally at its site of origin, or remotely at another site with lighter load. $\mathcal{PS}$, the problem solver, makes these decisions based on the job's characteristics as well as information about the past, current, and projected loads at various sites. $\mathcal{E}$, the learning environment, provides feedback by measuring the speedup (reduction in completion time w.r.t. local execution) of each job.

Different formulations of the load-balancing problem represent different degrees of realism. Abstract formulations assume that i) the background load at sites does not change outside the control of $\mathcal{PS}$ (no natural dynamics), and ii) $\mathcal{E}$ provides background knowledge relating migration decisions and measured speedup. More realistic formulations drop both of these assumptions. ■

Example 2. *Learning to Steer a Ship.* The performance task $\mathcal{P}$ is to steer a ship in a variety of (simulated) training scenarios, eventually getting as close as possible to a specified target position, which may be different for different scenarios. A scenario is described by the current coordinates of the ship relative to the target as well as current velocity vectors. Using these values, the problem solver $\mathcal{PS}$ must determine the direction and amount of turn at each decision point. In contrast to load balancing, the learning environment $\mathcal{E}$ provides $\mathcal{PS}$ with rudimentary operator-application rules and some general domain knowledge. ■

Example 3. *Learning to Balance a Pole.* The controller $\mathcal{PS}$ aims to balance a pole that is fixed at one end to a mobile cart with one (rotational) degree of freedom. At any decision point, the cart can be moved either left or right by applying a fixed amount of force. The cart is mounted on a rail of fixed length and is constrained to not go off the ends of the rail.

This performance task $\mathcal{P}$ exemplifies the class of control problems [20, 21]. Other examples in this class include: i) *regulation*, in which the objective is to keep the external environment close to a "desired state," ii) *tracking*, in which the objective is to make certain output variables follow the same sequence as their corresponding input variables, iii) *optimal-path problems*, in which the objective is to get the external environment in a desired state at a desired time, and iv) *minimum-time*

Table 2: Components of a performance task.

Component	Description
Variables	External inputs (*problem variables*) and problem-solving actions (*decision variables*).
Objectives	Goals of problem solving and strategy learning, *i.e.*, functions to be optimized or conditions to be achieved.
Constraints	Conditions that should not be violated by solutions.

optimal control, in which the objective is to get the external environment in a desired state in minimum time.

Control problems can be either knowledge-intensive or knowledge-lean; the latter are classified under *adaptive control*. Knowledge-intensive versions assume closed-form objective functions and complete knowledge of state changes (modulo noise) resulting from operator application; these tend to use static strategies called open-loop control. Knowledge-lean versions assume only that objective-function values can be sampled at each of the different states of $\mathcal{E}$; these tend to use dynamic strategies also known as closed-loop control. Controllers usually employ state-space representations that satisfy the Markov property. ■

Example 4. *Learning Strategies for the Towers of Hanoi Problem.* The scenario consists of three towers, of which the first is surrounded by rings whose diameter decreases from bottom to top. The objective is to move all the rings, in the same order, onto the third tower. The only operator available is one of lifting a ring from the top of one tower and placing it on top of another. The constraints are that a ring can never be placed over a smaller ring.

This performance task exemplifies the class of *symbolic problems* that have been studied in cognitive science and AI [22, 6]. These problems feature input variables that do not vary outside the control of $\mathcal{PS}$. ■

The remainder of this section is devoted to a detailed analysis of the anatomy of strategy-learning problems: the first subsection examines performance tasks; the second, problem solvers; and the third, learning environments. The final subsection isolates the key issues in strategy learning and relates them to the anatomy of strategy-learning problems.

2.1 Performance Task

A performance task $\mathcal{P}$ can be specified in terms of its objectives, constraints, and variables (Table 2). Objectives may be specified either explicitly (as functions to be optimized) or implicitly (using measured quantities indicating objective-function values of certain states); the former are called well-defined objectives and the latter, ill-defined. Similarly, constraints may be defined either explicitly (as truth-valued

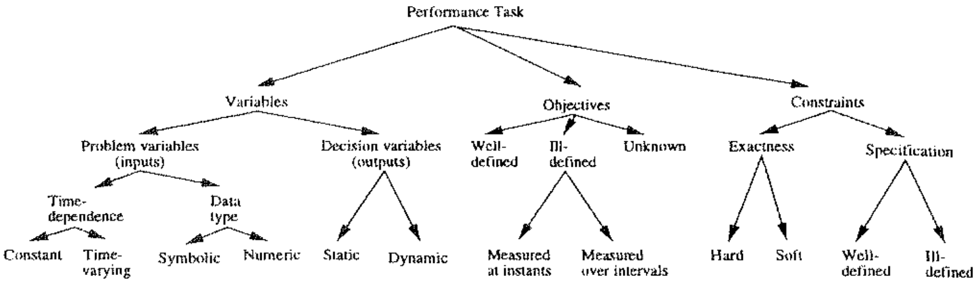


Figure 3: Attributes for classifying performance tasks.

expressions of variables) or implicitly (using measured quantities indicating feasibility); the former are called well-defined constraints and the latter, ill-defined.

The variables of a performance task serve two purposes: representation and modification of external state. The space of all possible assignments of values to all of the variables of a performance task is called the *state space* of that task. An assignment of values to the set of decision variables is called a *solution*. States satisfying all the constraints of a performance task define the space of *feasible solutions*. For optimization objectives, the feasible states that satisfy the optimization criterion define the set of *optimal solutions*. In general, states satisfying both the objectives and constraints are called *goal states*. Sometimes, the term solution also denotes a state-space path from the initial state for a given instance to some goal state of that instance.

The objectives and constraints, respectively, of a performance task are ways to measure its quality and feasibility; together, they constitute a *performance standard* [23] against which strategies may be judged. Problem solvers may either add new constraints (such as deadlines or limits on memory usage) or refine the objective function by trading between quality and complexity of solutions. Even so, the objectives of performance tasks are the prime drivers of problem solving and strategy learning.

Various properties of variables, objectives, and constraints determine the complexity of strategy learning; these are, therefore, useful attributes for classifying strategy-learning problems (Figure 3).

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Each site of a distributed system has private resources, such as CPU and memory. All sites share access to common resources such as networks and disks. The completion time of a job at a particular site depends upon the level of utilization of that site's private resources as well as on the level of utilization of common resources: the greater is the competition for these resources due to background load, the longer a job takes to complete. In addition, the basic service requirements of a job determine how much resource it would need even if there were no competition from other jobs.

The variables of the load-balancing problem include *status variables* (utilization levels of various resources, such as CPU, memory, disk, and network), *job descriptors* (absolute and/or relative amounts of service required by a job from the various resources), and *placement variables* (for each incoming job, the site at which it will be executed).

Different load-balancing systems target different objectives: maximum throughput (number of jobs completed per unit time); minimum average completion time (over all jobs); or minimum completion time per job. All these objectives are functions of completion time. Without any background knowledge of the relationship between variables and completion time, all of them are ill-defined functions.

Constraints may include *precedence constraints* (e.g., some jobs cannot start before certain others finish), *size constraints* (e.g., requirements on physical memory), and *placement constraints* (e.g., no more than one foreign job per site at any time). These are examples of well-defined constraints. Other constraints, such as preference for placement of communicating jobs at the same site, are ill-defined when the communication characteristics of jobs are not known ahead of time. ■

2.1.1 Variables

Those variables of a performance task that represent the problem solver's input parameters are called *problem variables*, and those representing the problem solver's output parameters, *decision variables*. An assignment of values to problem variables defines an *instance* of a performance task. An assignment of values to decision variables defines a solution. The values of problem variables are affected by changes in the state of the external environment. The values of decision variables are changed by the problem solver when its strategy is applied to particular problem instances.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Status variables and job descriptors are the problem variables, and job placements are the decision variables. A load-balancing system uses information about resource utilization and job characteristics in deciding where incoming jobs are placed. ■

Temporal Dependence of Problem Variables. Problem variables are said to exhibit temporal dependence when their values vary with time. Without temporal dependence, the external state varies only under the problem solver's control; future states are relatively easy to predict, enumerate, and evaluate. With temporal dependence, however, the external state may continue to evolve even after an action has been taken. The key issue is, therefore, the prediction of future states defined as Issue 1 in Table 3.

Example 2 (cont'd). *Learning to Steer a Ship.* Ships tend to have a large momentum. If no steering operations are taken, the current speed and direction of the ship (and water current, if significant) determine the ship's course. Given qualitative rules describing the ship's dynamics, steering actions are chosen only after considering the (predicted) states of the ship under the combined effects of the causal and natural dynamics of steering. ■

Table 3: A summary of all the issues in strategy learning.

Issue	Details
1: Prediction of Future States	Problem solvers and strategy-learning systems choose strategies based on the quality of external states produced. An important issue is the prediction of time-varying problem variables that requires consideration of natural dynamics (the laws governing time-dependent behavior). Certain learning environments provide qualitative laws describing the natural dynamics as background knowledge; in such scenarios, the problem of predicting future states is one of reasoning with qualitative temporal models [24]. In others, the model of temporal variation must be induced by the learning system based on repeated observation of state sequences; in such cases, prediction of future states is a problem of learning to predict time series [25].
2: Generalization across Instances of the same Performance Task	Except for the most trivial problems, it is impossible to expose a strategy-learning system to all possible instances of a performance task. Even so, the strategies developed by a learning system are expected to work on 'similar' (hitherto unseen) instances. The issue of generalization concerns how well the strategies learned perform on such new instances. For performance tasks with numeric variables, generalization is spontaneous because well-behaved decision functions yield similar outputs for similar inputs. However, with symbolic variables, the learning system must generalize explicitly; *i.e.*, it must determine the exact set of instances that can be solved using a solution developed for one particular instance.
3: Dynamic Decision Making	In dynamic decision problems, interactions among decisions need to be considered. Dynamic decision variables require causal models for representing the interdependence between decisions and states. Briefly, a *causal model* is defined as a set of rules for determining the new state given an old state and a decision. It must be stressed that for tasks with time-varying problem variables, the new state has a causal component (due to decision-making) as well as a temporal component (due to the natural dynamics of the external state).
4: Standard-of-Comparison Problem	This problem, first recognized by Ackley [26], concerns the method of assessing feedback. If the objective function of a performance task is well-defined and the Markov property is satisfied, then dynamic programming provides a way to choose the optimal alternative at each decision point, thereby defining an optimal strategy. Other decisions can be evaluated with respect to this optimum. If, however, the objective function is ill-defined, it is difficult to assess solution quality in absolute terms, and alternative operators can only be evaluated relative to each other.
5: Learning an Objective Function	When an objective function is ill-posed, information about the goals of problem solving is implicit in the feedback associated with each state. Since feedback for a state is generated only *after* the state has been traversed, learning the objective function is more difficult. Two cases need to be considered. First, when states can be evaluated independently, one can tabulate evaluations as <state,feedback> pairs. The problem of making the learning problem well-posed then reduces to one of fitting a function to the tabulated data. Second, when measured objective-function values depend on several states, one can regress either a simple function upon the current and past values of problem variables, or an autoregressive (recursively defined) function on just the current values.

Table 3 (cont'd): A summary of all the issues in strategy learning.

Issue	Details
6: Learning while Searching	This is also known as within-trial learning: the nondeterminism in later stages of search is reduced using information about states visited in earlier stages. It is especially useful for learning stochastic strategies which, depending on the amount of prior knowledge available, may start with a random search and eventually converge to an almost deterministic search [27]. Learning while searching is also useful for problem solvers that employ deterministic strategies to expand large search spaces [28, 29] For learning with asynchronous feedback, which precludes identification of clean learning episodes, as well as for learning in changing environments, learning while searching is the only tractable way to learn. Learning while searching requires the learning algorithm to have low overhead because learning occurs at each decision point.
7: Constraint Handling	Constraints limit the solution space for decision variables on the one hand, and impose restrictions on generalization by the learning system on the other. The strategies generated by the learning system must not recommend operators that lead to infeasible states.
8: Violation of Markov Property	This issue entails the incorporation of past states and decisions into the current decision point: a history of past decisions needs to be maintained for both decision making and TCA. Corollary issues, such as how to 'forget' old or unimportant decisions, must also be addressed [30]. The problem is particularly acute for tasks with time-varying problem variables because past states and decisions may carry information useful at future decision points.
9: Storing Past Decisions	There are two related problems: managing the storage of the state vectors leading to the final state, and optimizing the time for distributing credit among them. The first problem in maintaining such a history is that of size. Not all the information in a state vector is relevant, nor is it feasible to go through the entire history each time a feedback signal is received. Determination of the past information to be retained then becomes important. Even with non-Markovian state spaces, and especially with time-varying problem variables, the effects of decisions become insignificantly small after a certain time interval; such decisions should be 'forgotten' or deleted from the history. The second problem concerns the extraction of relevant information from past history so that when feedback becomes available, it can be distributed among decisions in proportion to their contribution to the state(s) being evaluated by the current feedback signal.
10: Managing General Objectives	With general problem solving, learning systems are expected to learn strategies whose "desired state" may vary from one problem-solving scenario to another. This requires representation schemes that bring out the internal structure of objectives as well as the techniques for achieving general goals. The learning system may need to use general-purpose reasoning techniques [31, 32]. In this case, the procedural component of the knowledge stored should be reduced to a minimum; such reduction in procedural knowledge may slow down learning as well as problem solving. A more important consequence is that the learning system may need to acquire different strategies for meeting different objectives. This raises the need for efficient indexing of strategies by goals [33].

Table 3 (cont'd): A summary of all the issues in strategy learning.

Issue	**Details**
11: Handling Structured Solutions	For generating structured solutions, static strategies are preferred over dynamic ones because static strategies can simultaneously consider multiple (interdependent) decision points. Static strategies are feasible in knowledge-rich problems in which the strategy-learning system exploits the structure of the solutions already tested in order to find solutions for problems not seen before. Such learning requires substantial deductive reasoning as well as inductive generalization of solutions. Static strategies are not possible in knowledge-lean applications because there is not enough prior knowledge to guide the design of such strategies. In this case, new knowledge acquired during learning must be incorporated into new strategies in the system, causing considerable overhead during learning.
12: Nondeterminism	The strategy learner must be designed so that it can discover heuristic rules for choosing the 'best' alternative without having to explore all of them. In knowledge-lean environments, alternatives cannot be compared a priori because there is insufficient information. Moreover, the exploration of an alternative may have an irreversible side-effect on the external state, especially for tasks having time-varying problem variables. In this case, the learning system may sustain nondeterminism by introducing a controlled element of randomness in decision making. Only one randomly selected alternative is explored on each visit to a state; but several different alternatives may be explored if each state is visited several times. This is achieved using stochastic strategies, which are popular in strategy-learning systems for knowledge-lean environments [27, 34].
13: Structural Credit Assignment (SCA)	SCA deals with methods for apportioning feedback $\mathcal{F}$ to elements of the decision process. These methods depend on the 'structure' of the decision process, and whether $\mathcal{F}$ is prescriptive or evaluative. It is often more difficult when the decision process is procedural and the feedback, evaluative.
14: Temporal Credit Assignment (TCA)	TCA is the first stage in the assimilation of feedback and precedes SCA during learning. It divides up feedback $\mathcal{F}$ between current and past decisions. Methods for TCA depends on whether the state space is Markovian. Non-Markovian representations and direct operations often require more complex TCA procedures.
15: Predicting Future Feedback	Predicting future feedback is important when feedback is delayed; here, the learning system needs to estimate future feedback in order to determine the magnitude and direction of parameter modification. When the environment produces immediate prescriptive feedback, the learning system simply attempts to reduce the error between the observed and the desired values of decision variables. However, when the feedback is evaluative, the learning system must learn to predict the externally generated feedback signals using its own internal state. Especially when feedback is temporally global, or when the objective function is ill-defined and measured over intervals, it is not obvious which states will lead to better feedback. While prior prediction of future feedback is useful for decision making, a posteriori association of feedback and states (as in the TCA problem) is useful for learning.

Numeric versus Symbolic Problem Variables. Problem variables can be either numeric or symbolic. For performance tasks having numeric variables only, the degree of similarity between different instances can be defined using distance functions in the input space. Symbolic values, on the other hand, often exhibit internal structure, which precludes numerical characterization of similarity between instances. The issue involved is stated as Issue 2 in Table 3.

Static versus Dynamic Decision Variables. The goals of problem solving are specified as functions and formulae of decision variables. A problem solver *instantiates* (assigns values to) decision variables during decision making. The final assignment of values constitutes a solution state. Frequently, decision variables represent actions and their parameters. Decision variables whose instantiation affects future inputs to a problem solver (*i.e.*, future values of problem variables) are called dynamic decision variables; and those whose instantiation does not affect future inputs, static decision variables. Temporal dependence is the only source of variation for strategy-learning problems having static decision variables (called *static decision problems*); in this case, the context of strategy learning can be limited to one decision point at a time. The more difficult issue is on *dynamic decision problems* where the external state evolves under the influence of the problem solver's actions. The related issue is summarized as Issue 3 in Table 3.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Usually, $\mathcal{PS}$ employs a *dynamic load-balancing* strategy that decides placements based on values of status variables. When a job is migrated from one site to another, its net effects are reduced load on resources local to the originating site and increased load on resources local to the remote site. Therefore, status variables evolve under the effect of placements, making this a dynamic decision problem.

However, there exist formulations of this problem in which all the jobs to be placed are available at the outset, and placements do not depend on status variables. Such strategies are appropriately called *static load-balancing strategies.* In these, job descriptors are the only problem variables and do not evolve under the effect of placements, making this a static decision problem. The context of strategy learning includes only one decision point: the one at the initial state, although each decision (of simultaneously placing all the jobs) tends to be complex. ■

2.1.2 Objectives

Figure 3 shows that various degrees of precision are possible in specifying the objectives of a performance task. Traditionally, problem solvers have attempted only performance tasks having closed-form objective functions. Realistic applications, however, frequently involve ill-posed tasks whose objective functions are either unknown or empirically determined.

Well-defined, Ill-defined, and Unknown Objectives. Objectives are said to be well-defined when a closed-form objective function of problem variables is (either explicitly or implicitly) specified, and ill-defined when such a specification is unavailable but states can be evaluated either individually or collectively. When the objective function is unknown, the problem solver must use prior knowledge to assume either a well-defined or an ill-defined objective function. In the case of ill-defined objective functions, the measured objective-function values may evaluate states either individually or collectively. In the latter case, evaluations are not available for every state and, when available, measure the overall quality of a sequence of successive states.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Objectives such as equalizing the load across all sites are well-defined provided 'load" itself is a well-defined entity. In abstract formulations of the problem, popular with queuing theorists, CPU is the only resource of contention, and the level of CPU utilization is considered the sole determinant of the overall load. The objective is well defined because load can be defined easily in terms of the available inputs (CPU utilization).

Usually, the objectives of load balancing are defined in terms of job completion times. Without extensive knowledge of hardware architecture and the operating system's scheduling policy, completion-time-based objective functions are ill-defined. Automata theorists [35] choose to formulate completion time as a function of only the current values of status variables; this results in an ill-defined objective function whose values can be measured for states individually. In this case, a deterministic evaluation function can be learned by regressing job completion time onto the values of status variables at the time of starting the job.

In reality, the completion time of a job depends on past, current, and future values of status variables [36], which makes it an ill-defined function whose values evaluate multiple states. If non-recursive models — such as MA [37] — are used in regression, then the window of past status variables to consider must be determined first. If, on the other hand, recursive regression models — such as ARMA and ARIMA [37] — are used, then the coefficients of the model must be determined using the relatively complex time-series regression techniques.

A number of alternative objectives are possible for the load-balancing problem. For instance, if *throughput* (number of jobs processed per unit time) is to be maximized, then it does not matter whether or not specific jobs complete sooner with load balancing than without it. This objective is valid for users who submit jobs in batches (groups), and only the performance of the batch is relevant. On the other hand, when scheduling independent jobs, it may be important to have a high probability of speeding up each individual job. In this case, good performance on large jobs cannot compensate for poor performance on small jobs. A learning system starting without a specific objective must compare alternative proposals and choose the one whose predicted value matches the actual feedback from the user. ■

For performance tasks with ill-defined objective functions, the goals of learning and problem solving are not clear at the outset; they must be inferred using either prior knowledge or goal-related information implicit in the feedback. For problems with non-reactive learning environments and ill-defined objectives (Class 8 of Figure 1), feedback must be generated internally by the learning system.

The following three issues arise in learning strategies for performance tasks with ill-defined objective functions: standard-of-comparison problem (Issue 4), learning an objective function (Issue 5), and learning while searching (Issue 6). We define these issues in Table 3 and illustrate them by the following examples.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* To illustrate Issue 4, consider the load-balancing problem that has ill-defined objectives. To solve the standard-of-comparison problem in the evaluation of load-balancing strategies, the case with no load balancing is often used as a point of reference. This approach requires that two sets of experiments be performed using exactly the same set of jobs and loading conditions: once with, and once without, load balancing [38].

To illustrate Issue 5, note that completion time is not a well-defined function of status variables alone, and job descriptors are rarely available when decisions are made. The completion-time-based objective functions are ill-defined and can be evaluated only over state sequences. When queuing models of computers and jobs are available and applicable, objective functions are well-defined, and the queue sizes on various resources of contention [39, 40] can be shown to be useful for predicting completion time. In practice, the use of these models is questionable, and objective functions need to be modeled from data relating completion times of jobs to load conditions at the time of decision making. ■

Example 3 (cont'd). *Learning to Balance a Pole.* As the learning system gets better at balancing, the episodes become longer with no external feedback. To limit memory usage, the learning system must learn within trials. One way to do this is for the learning system to predict future feedback, and use its own predictions in place of the real feedback. Whenever external feedback does become available, it can be used to train the internal feedback-generation mechanism. ■

2.1.3 Constraints

The constraints of a performance task determine its space of feasible solutions over which optimization is performed. They may either be explicitly specified, or be built into the problem solver's strategy, or be part of the objective function. Constraints can be classified on the basis of their exactness (as hard or soft), as well as their specification (as well-defined or ill-defined) (Figure 3).

Hard versus Soft Constraints. Hard constraints impose sharp boundaries on the state space, demarcating feasible solutions from infeasible ones. They curtail syntactic generalization (syntactically similar problems having similar solutions) and are usually enforced by a *move generator* in the problem solver. Problems with hard constraints are sometimes solved by first solving relaxed versions of the original problems in order to obtain an approximate solution, which is then used as an initial state for an exact solution. On the other hand, problems with soft constraints are associated with a large space of feasible solutions. These constraints are usually transformed into penalty terms that are added to the objective function.

Well-defined versus Ill-defined Constraints. Well-defined constraints are defined as truth-valued functions of a performance task's variables; hence, one can determine whether a solution is feasible or not by testing whether it satisfies the constraints. In contrast, ill-defined constraints are either unknown or too complex to be modeled as functions of problem and decision variables. As a result, the constraints cannot be formulated as truth-valued functions. In some cases, ill-defined constraints become well-defined during the course of problem solving and are incorporated in the problem solver. For instance, if a robot is attempting to fit a projection on one part into a slot in another but does not have geometric models of the two parts, then it can discover infeasible moves by testing whether the move is feasible and by hypothesizing a model of infeasible regions. If the robot knew that all slots are square in shape and aligned parallel to some known coordinate system, then it may find the exact coordinates of the slot after a few trials.

The key issue on constraint handling (Issue 7 in Table 3) is related to how constraints are handled by $\mathcal{PS}$ that are to be learned by the learning system.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Graph-theoretic formulations of load balancing take into account explicit precedence and placement constraints both of which are hard, well-defined constraints. System engineers, on the other hand, either ignore these dependences or enforce them procedurally in the problem solver. Size constraints in load balancing are examples of soft, ill-defined constraints because the memory requirements of a job are usually data-dependent and cannot be modeled analytically. Too large a job for a small physical memory causes only a gradual degradation in performance. Thus, there is no sharp boundary between feasible and infeasible states. Because of such 'softness,' size constraints are often expressed as preferences, becoming part of the objective function. ■

Example 4 (cont'd). *Learning Strategies for the Towers of Hanoi Problem.* This problem has one significant hard constraint, namely, that no disk can ever be placed on another of smaller diameter. When generalizing from solutions to specific subgoals to strategies for achieving more general goals, the learning system must ensure that the strategies it learns do not violate this constraint. ■

Table 4 illustrates the structure of $\mathcal{P}$ for the four strategy-learning problems described in the beginning of Section 2.

2.2 Problem Solver

Problem solvers employ *representation schemes* to internally represent problems and solutions; they also have a repertoire of *operators* which are used for transforming initial states into goal states. (A sequence or a partial order on the set of operators constitutes a solution.) The problem solver uses parameters that may be tuned via learning as well as others that are constrained in its design. In this section, we identify those aspects of problem solvers' representations, operators, and strategies that influence the design of strategy-learning systems. The components of problem solvers and their properties are shown in Figure 4.

Table 4: Examples of performance tasks in strategy learning.

Example	**Performance Task $\mathcal{P}$**	**Comments**
Load Balancing	To schedule incoming jobs	Problem variables (indicators of background workload, job characteristics) are numeric and time-varying; decision variables (placements), dynamic.
Ship Steering	To decide when and how much to turn	Problem variables (position and velocity) are time-varying; and the decision variables (amount of turn), dynamic. The objective function (navigating the ship along a trajectory) is ill-defined, measurable over intervals.
Pole Balancing	To balance an inverted pendulum	The problem variables (the pole's angle of inclination and angular velocity, and the cart's position and linear velocity) are numeric time-varying; and the decision variables (direction of applied force), dynamic. The objective is ill-defined, measurable over instants.
Towers of Hanoi	To achieve a desired configuration of disks on the towers	The problem variables (specifying one of the three towers each disk is on) are symbolic and constant; and the decision variables (which disk to move where), dynamic. The objective (to achieve a desired configuration) is well-defined; and the constraints (no disk can be placed over another of smaller diameter), exact.

2.2.1 Representation

States and objective functions are significant components of strategy-learning problems; their representation determines the type of strategy-learning technique used. Following are the key properties of states and objective functions that affect strategy learning.

Markovian versus Non-Markovian State Space. If each state carries enough information to permit optimal decision making without consideration of past states or past decisions, then the state space is called Markovian. Markov property asserts that the future behavior of a system is not affected by past states, given the current state [41]. Its manifestation in decision making is the *path-independence* axiom: the optimal decision in a state does not depend on the state-space path leading to that state [42]. For such state spaces, past states and past decisions do not influence current decisions.

If the state space is non-Markovian, then a history of past states and/or past decisions needs to be maintained for use in making future decisions. Often in the former case, large state vectors may be needed to capture all of the useful information in one state. When the number of problem variables is large, one may choose to track only a few significant variables rather than retain the entire state vector: such representations will be inherently non-Markovian.

Violation of Markov property is especially easy to verify for systems having time-varying problem variables. One can study the partial autocorrelations [37] of the time series generated by problem variables: nonzero correlations at lags

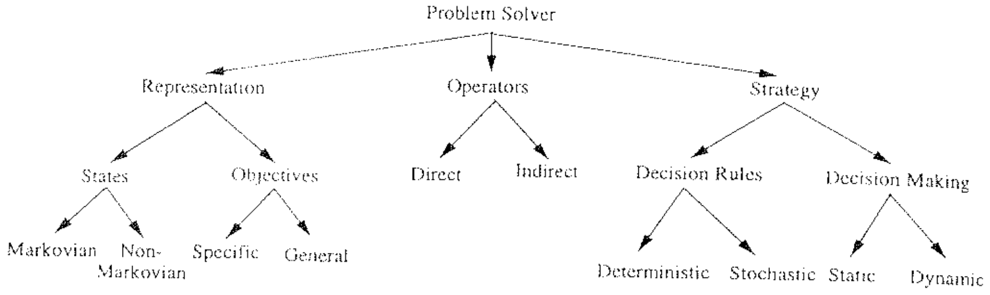


Figure 4: Components of problem solvers and their properties.

greater than one are the simplest evidence for the violation of Markov property. The corresponding issue (Issue 8) is described in Table 3.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* The violation of Markov property is clearly illustrated by significant partial autocorrelations at lags greater than 2 observed in the time series generated by the status variables [36]. ■

The violation of Markov property and the consequent importance of past decisions during TCA entail the storage of past decisions and possibly even the states in which those decisions were taken. The related problems on storing past decisions are summarized as Issue 9 in Table 3.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Since this problem violates Markov property, past states and/or decisions need to be stored. The effects of past states on future decisions are captured in decision metrics (called *load averages*) that are used in decision rules instead of status variables. The formulae for computing load averages can either be user supplied [43] or be induced from completion-time measurements [44]. ■

General versus Specific Objectives. A problem solver may be either *general-purpose* or *specialized*, depending on whether it can represent and solve a class of problems rather than a single problem. General problem solvers need to index their strategies by the structure and the content of their objective functions. Unlike specialized problem solvers, the ability to represent and manipulate explicitly represented objectives is paramount in this case. Specialized problem solvers are common in the literature on control [20], whereas generalized problem solvers are common in the literature on planning and problem solving [45]. Strategy-learning systems for general-purpose problem solving face the problem of *generalizing across tasks*; that is, the experience from one problem may need to be generalized to a different problem. For specialized problem solvers, strategy learning merely warrants *generalizing across instances* of the same problem (Issue 10 – managing general objectives – in Table 3).

Direct versus Indirect Operators. Problem solvers transform their internal and external states by applying operators. Operators can be either direct or indirect. *Direct operators* act independently on the objective function without interfering with other instances of operator application. The effects of direct operators combine in simple ways, satisfying criteria such as additivity and superposition [46]. On the other hand, the effects of *indirect operators* combine in complex ways, such as through causal chains or AND-OR graphs [47]. In this sense, strategies involving indirect operators generate solutions having more 'structure.' Numerous examples exist, especially in planning [48], of problems that require such indirect problem-solving capability. Strategies using indirect operators warrant complex strategy-modification techniques.

Indirect operators are used in structured solutions: a solution is structured when a number of indirect operators contribute to the reward state(s) of the external environment. Indirect operators, in conjunction with hard constraints and symbolic problem variables, characterize some of the most complex learning problems. An important issue is, therefore, the design of strategies for generating structured solutions under various environments. (Issue 11 in Table 3).

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Placement of independent tasks is a direct operator because its effect on the external environment can be completely captured by changes in the values of the status variables. Different instantiations of this operator interact in simple ways, reducing load at the source of a migrating job and increasing it at the destination. Thus, balancing tasks without precedence constraints result in unstructured solutions. On the other hand, the placement of dependent tasks requires (indirect) operators for matching, sorting, and assignment, which combine into structured solutions. ■

2.2.2 Problem-Solving Strategies

A problem solver should have an internal bias towards certain preferred modes of problem solving. One without such preferences would be horribly inefficient for most practical applications, despite its tremendous generality. Another reason for bias is that expert knowledge may be available only in the form of time-tested procedures. For efficiency and practicality, such procedures should be built into problem solvers as skeletal strategies, and subsequently refined by strategy learning.

Strategies can be classified, based on the type of decision rules employed, into deterministic and stochastic. Further, one can distinguish between static and dynamic strategies on the basis of when the problem-solving decisions are made. These distinctions are explained below.

Deterministic versus Stochastic Strategies. A deterministic strategy always recommends the same operator for an external state in a given context, no matter how many times this state recurs. Stochastic strategies, on the other hand, explore multiple alternative decisions at those decision points where a choice cannot be made *a priori*. Extra knowledge for guiding such a search may take the form of decision rules for assessing either the probability that a particular alternative will be explored, or the order in which alternatives will be explored. Stochastic strategies have the potential for producing better solutions at the cost of extra computations.

Several problems discussed earlier — standard-of-comparison and learning-while-searching — are related to the basic problem of nondeterminism in problem solving. When a problem solver is forced to choose between alternatives, it must attempt to reduce the nondeterminism by limiting the number of alternatives actually explored. The challenge of nondeterminism is summarized as Issue 12 in Table 3.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Instead of instantiating the decision variable directly, the controller $\mathcal{PS}$ computes the probabilities of the left and the right moves at each decision point [49]. The actual assignment of a value to the decision variable is performed randomly using these probabilities. This approach is easily extended to the case with multiple alternative actions at each decision point [50]. As the probabilities of selection become more and more biased in favor of good alternatives, nondeterminism is reduced. ■

Static versus Dynamic Strategies. While certain strategies make decisions based on the specific state associated with each decision point, others recommend a series of decisions based on just the initial state. The former are called dynamic strategies; and the latter, static strategies. Static strategies are faster but inappropriate for problems whose variables cannot be predicted accurately or efficiently a priori. Designing static strategies requires a complete and accurate model of the external environment; dynamic strategies, because they make decisions about one or a few actions, require less prior knowledge and are simpler to design. However, dynamic strategies are computationally more expensive to apply than static ones because they warrant run-time information gathering and inference.

Strategies generated by conventional planning methods of AI [51] are static, whereas those generated by conventional dynamic programming methods of control and optimization [42] are dynamic. There exist numerous variants of conventional planning techniques that use dynamic strategies; these are called reactive planning methods [52, 53, 54, 55, 56, 57] in AI.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Two types of problem solvers represent, respectively, the analytical and software-based approaches to the load-balancing problem. Both are designed for specific objectives and employ Markovian representations, using only the current state in making load-balancing decisions. Because the status variables do not directly permit such representation, these problem solvers use 'load indices,' which are abstract variables computed as moving averages of computation load. Often, the coefficients of moving averages are chosen ad hoc. An alternative considered in some of our recent work [36] is to employ non-Markovian representations and *learn* the relative importance of past and current values of status variables for predicting speed-ups of tasks.

The problem solvers designed for handling precedence constraints often employ *indirect operators* in which a number of different operators contribute to the reward state(s) of the external environment. Solutions in this case are said to be *struc-*

Table 5: Examples of problem solvers in strategy-learning tasks.

Example	Problem Solver $\mathcal{PS}$	Comments
Load Balancing	Process migration software	Representation of states is non-Markovian; of objectives, specific. The operators (process migration) are direct. Practical strategies employ dynamic decision making and either deterministic or stochastic decision rules.
Ship Steering	A program for navigation	Representation of states is non-Markovian; of objectives, general. The operators (turning actions) are direct. The strategy is static, and the decision rules, deterministic.
Pole Balancing	A controller to apply a fixed force left or right	The controller employs Markovian representations for states, and works for a specific objective. The operators (applying the force) are direct. The controller uses stochastic strategies.
Towers of Hanoi	A program for partitioning objectives and for searching among moves	The problem solver employs a Markovian representation of states and is able to handle general-purpose goals because multiple goals are generated for each problem instance. Its operators (moving disks) are indirect, and its strategies, deterministic and static.

tured. Others generally use the simpler direct operators such as sending a job to a remote site and accepting a job sent by a remote site. Analytical solutions often employ stochastic strategies [35, 58, 59], although deterministic strategies are more common [60, 61].

Dynamic load balancing strategies take into account the *load average* at each site in deciding where to send the next job, whereas static strategies schedule jobs according to a predetermined criterion unaffected by the dynamic variations of workloads. While static strategies avoid frequent communication of status information among sites, they fail to exploit dynamic imbalances in load whose occurrence cannot be predicted ahead of time. ■

Table 5 illustrates the structure of $\mathcal{PS}$ for the four strategy-learning problems described earlier.

2.3 Learning Environment

The interface between a strategy-learning system and the external world is called the learning environment. Environments that generate feedback are called *reactive environments*. These include human trainers and/or programs that generate feedback for the learning system, as well as other sources external to the learning system that provide prior knowledge relating problem variables and decision variables to feedback. If the environment is not reactive but the objective function is measurable (Figure 1), then the measurements can be used in feedback generation. In this case, certain additional issues such as the standard-of-comparison problem need to be addressed. The components of a learning environment are shown in Figure 5.

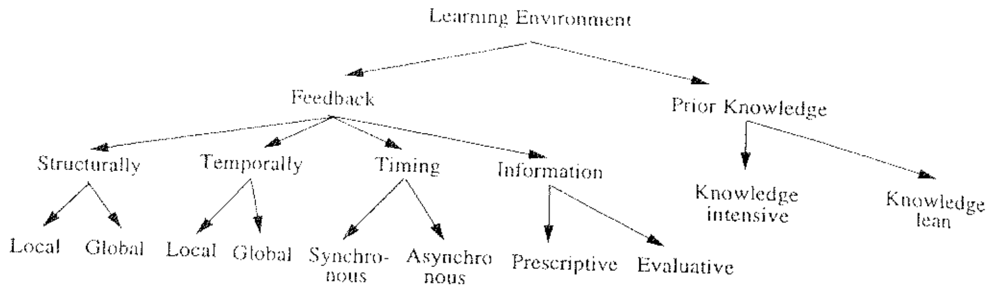


Figure 5: Components of a learning environment.

2.3.1 Feedback

The nature of feedback is the single most important determinant of the form and complexity of strategy-learning algorithms. Feedback may be a corrective error signal or a scalar evaluation signal, covering either one or more decision points, and generated either periodically or intermittently. Feedback can even be generated internally in the learning system when little or no external feedback is available.

Translation of environmental feedback (F) into strategy modifications is called credit assignment. It is a problem in inverse modeling of both the environment and the problem solver: the learning system attempts to determine what changes in the decision process will bring about desirable feedback on similar instances in the future. Different types of feedback require different schemes for credit assignment. Since credit assignment is a key component of any learning algorithm, different types of feedback require different kinds of learning algorithms.

Prescriptive versus Evaluative Feedback. Feedback signals carry certain explicit and implicit information useful for altering the behavior of a problem solver. Signals that carry more explicit information require simpler learning rules but a more informed source than those that carry more implicit information. Based on the amount of explicit information, one can distinguish between prescriptive and evaluative feedback.

Prescriptive feedback carries explicit information about the desired operators and/or states; the learning system can use it for computing an error signal to be minimized via strategy modification. However, generating such feedback requires a *teacher* who knows what the correct outcome should be. Learning from a teacher is called *supervised learning* [62].

Example 2 (cont'd). *Learning to Steer a Ship.* Since the desired course of the ship is specified, this is an example of prescriptive feedback. At every decision point, the problem solver knows how far off the desired course it is. ■

Evaluative feedback carries only implicit information about the desired behavior but explicit evaluation of the observed behavior. Generating such feedback requires only a *critic* [63] who has some prior knowledge of the objective function and can assess the goodness of external states or sequences thereof. Scalar evaluative feedback signals are called *reinforcements* [9] and learning from such signals, *reinforcement learning*.

Example 3 (cont'd). *Learning to Balance a Pole.* The learning environment generates (negative) feedback only when the pole falls down; otherwise, no feedback is generated. Such a feedback specifies only how well a problem solver is performing but not what the desired external states are. This is an example of evaluative feedback. ■

Structural Locality of Feedback. The goal of learning is to modify either the decision rules or certain parameters thereof. Many rules or parameters may be involved at each decision point; these may be evaluated either individually or collectively: if individually, feedback is termed structurally local; otherwise, global. Usually, feedback is structurally global (the external environment evaluates decisions but not individual rules), and the evaluation of individual rules is left to the learning system. Translation of structurally global feedback into a structurally local one is called *structural credit assignment* (SCA) [10]. Examples of structurally local feedback are rare: in some learning environments, a source of knowledge external to the learning system may separate the individual effects of different rules and then feed these back to the problem solver; in yet others, the rules might operate on independent aspects of a problem, each associated with its own feedback signal (refer to ICA — independent credit assignment — discussed in Section 3.2.1). The issue involved is stated as Issue 13 — structured credit assignment — in Table 3.

When the rules encoding a decision maker's strategy combine in predictable and well-defined ways at every decision point, one can associate a 'structure' with the decision process. For instance, a proof tree might be used for representing a symbolic decision process, as in explanation-based learning [13]. Structures such as proof trees are generated dynamically. Examples of static decision structures include feedforward neural networks, as in Anderson's pole-balancing system [49]. In either case, the goal of SCA is to translate the (temporally local but structurally global) feedback associated with a decision point into modifications associated with various parameters of the decision process.

When a problem solver encodes its strategy using procedures rather than rules, the 'structure' of the decision process is not obvious. Here, SCA must either be avoided entirely or be used to associatively correlate the values of decision-process parameters with average feedback. For instance, certain problem solvers for load balancing [64] use parameterized decision procedures rather than a rule base [61] to represent their strategies. In these, SCA is used merely to associate the average completion time over a set of test jobs with every tested set of parameter values; such SCA perform *selection* among alternative strategies rather than *modification* of an incumbent strategy.

When feedback $\mathcal{F}$ is prescriptive and strategy S used by $\mathcal{PS}$ deterministic, SCA is relatively easy: it involves reduction of the difference between the observed and the desired values of the decision variables. For instance, in problems of tracking and trajectory planning [65], as well as in the ship-steering problem discussed earlier (Example 2), the desired final state is explicitly known. On the other hand, when strategy S used by $\mathcal{PS}$ is stochastic, SCA involves estimation and optimization of the *probability* of producing the desired outcome. For instance, in load balancing (Example 1), if the learning system forms internal models of jobs using Markov chains, [66], then it must adjust the transition probabilities of its model to match those of the job being executed.

When $\mathcal{F}$ is evaluative, it is not clear what the desired values of the decision variables should be, nor are the direction and magnitude of parameter modifications obvious. In this case, SCA requires assumptions about how the feedback signals evaluate the current strategy. SCA can be simplified when the decision process involves only a small number (say two or three) of discrete outcomes [50]. For instance, one might compare the current evaluation against a moving average of past evaluations in order to determine whether the probability of the observed outcome should be increased or decreased [26]: if the difference is favorable (positive for maximization of evaluation), the probability of producing the observed outcome is made closer to 1, and that of the remaining outcomes reduced accordingly; and vice versa.

SCA with evaluative feedback always involves search among alternatives. Stochastic strategies [27, 50] perform such search implicitly, and deterministic strategies [67], explicitly. At each decision point, the decision process chooses an operator to apply. When the decision process selects the operator producing the best evaluation, the values of decision parameters are treated as positive examples; otherwise, negative examples. SCA attempts to create decision regions in the space of problem variables so that each region has associated with it the best expected outcome for instances falling in that region. In general, there is a tradeoff between exploration (via search) and testing (repeatedly applying the decision process to new instances in order to gain confidence in the quality of the moves selected by the decision process).

Temporal Locality of Feedback. The environment produces feedback in response to the problem solver's decisions. Feedback may evaluate decisions either individually or collectively: if collectively, several decision points may elapse before feedback becomes available. Such feedback is called *delayed feedback*; it contains the combined evaluation of several decisions. Delayed feedback explicitly evaluates the current state and implicitly evaluates past states and decisions, especially for dynamic decision problems. Based on the temporal properties of feedback, one can distinguish between temporally local and temporally global feedback.

Temporally local feedback applies to decisions individually. In solving a large and complex problem, a problem solver may make many decisions; temporal locality requires that the environment should produce an explicit reaction to every decision. The burden of disentangling the interdependences among decisions is on the environment rather than on the learning system. Therefore, systems that learn from temporally local feedback are easier to design than the ones that learn from temporally global feedback.

Temporally global feedback applies to decisions collectively. Resolution of interdependence of decisions shifts from the environment to the learning system, which must distribute the feedback between decisions using knowledge of cause-effect relationships between decisions and feedback. Translation of temporally global feedback into temporally local feedback is called *temporal credit assignment* [10] (Issue 14 in Table 3).

If the state space is Markovian, then $\mathcal{F}$ can be modeled as a function of only the current state; as a result, recent decisions (which directly contribute to the current state) are more eligible to receive feedback than past decisions (which contribute only indirectly through intervening decisions and states). In particular, with direct operators (which independently cause the evaluation), it suffices to have a *discount factor* (less than 1) for determining the relative importance of each decision with respect to its successors. For instance, if the discount factor is 0.9, then the decision immediately preceding the feedback signal will have a weight of 1, the one preceding it 0.9, the next one 0.81, and so on.

For non-Markovian representations and indirect operators [30], past decisions and/or states may need to be retained because they may influence feedback $\mathcal{F}$ independent of the current state. Ill-defined objective functions that can only be measured over intervals also require the retention of past states and/or decisions. Determination of the relative importance of successive decisions may involve more than just a simple discount factor. Instead, the interdependence between different decisions may need to be captured explicitly using dependence graphs.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* In load balancing, credit assignment is complicated by delayed evaluative feedback. As jobs complete, they are evaluated and feedback signals generated. TCA involves dividing such evaluations among the scheduling decisions made at various sites. The non-Markovian state space of load balancing [36] requires explicit solution of TCA. Simple discount factors are inappropriate because the completion time of a job is not predominantly affected by the latest placement decisions. ■

Example 3 (cont'd). *Learning to Balance a Pole.* Since the system state following a left/right move is completely determined by the knowledge of its current state and the proposed move, the state space is Markovian, and the interaction between states decays exponentially with respect to time [37]. Such exponential decays can be computed dynamically as discount factors [68]. Efficient procedures [69] proven to work for Markovian representations are known for this and related problems. ■

Example 4 (cont'd). *Learning Strategies for the Towers of Hanoi Problem.* The operators are indirect but the state space is Markovian. Feedback is delayed with respect to decisions; hence, TCA must be addressed. Due to the causal connections between decisions and states, solutions are structured. In this case, TCA separates essential states (those on the path from the initial to the desired state) from non-essential ones. SCA is then applied to generalize the reduced solution structure so it can be used for solving similar problem instances in the future [13]. ■

Synchronous versus Asynchronous Feedback. The timing of feedback dictates the ease with which a learning system can partition its data into *episodes* or *trials*. When feedback occurs at predictable stages of problem solving, learning is episodic. Both the SCA and TCA problems discussed above are simplified when feedback distribution is confined within episode boundaries. Such feedback is called synchronous. When the time of occurrence of feedback is not easily predictable, the learning system is responsible for bounding the scope of feedback distribution. Only a finite number of past decisions and/or states can be retained between successive occurrences of feedback, and the learning system is responsible for their storage and TCA. Hence, an important issue with asynchronous feedbacks is the prediction of times for future feedbacks (Issue 15 in Table 3).

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Status variables in load balancing vary with time; predicting their variation as well as their effects on future values of feedback are the central issues in learning load-balancing strategies. Prediction of feedback is complicated by the violation of Markov property. As the states evolve with time under their natural dynamics as well as under the influence of load-balancing decisions, more effort is spent on the tractable backward projection of credit assignment, and less on the forward projection of predicting future states or future feedback signals. ■

Example 3 (cont'd). *Learning to Balance a Pole.* This problem features delayed and evaluative feedback. The length of episodes grows with experience because the problem solver can keep the pole balanced longer. To continue to learn within episodes, the learning system needs to predict future feedback so that it can substitute its prediction in place of the (missing) immediate feedback for each decision. The key (for Markovian representations only) is to express the total error of prediction as a sum of differences between successive predictions [70]. ■

2.3.2 Prior Knowledge

During credit assignment, a strategy-learning system needs a ***world model*** to capture the relationships among states, decisions, and feedback signals. When such knowledge is not given, it must somehow be inferred before/during credit assignment. Environments that provide an explicit world model to the learning system are called knowledge-rich; those that do not, knowledge-lean.

In knowledge-rich environments, credit assignment is a deductive process. The learning system can explicitly construct proof trees or other computational structures relating decisions and feedback. In knowledge-lean environments, the learner is forced to induce a world model from the states observed between making a decision and receiving feedback. Interleaving such induction with problem solving and strategy modification places extra burden on the learning system.

In knowledge-rich environments, the uncertainty about the future states, given the current state and action, is much less than that in knowledge-lean ones. Strategy-

Table 6: Examples of environments for strategy learning.

Example	Learning Environment $\mathcal{E}$	Comments
Load Balancing	Measurements and models of completion time	Feedback is a function of measured completion time. It is delayed, evaluative, asynchronous, and structurally global. Effect of placement decisions on the external state is unknown, as well as models of tasks and their inter-arrival times.
Ship Steering	Measurement of target displacement and models of steering actions	Feedback (error between actual and desired trajectories) is structurally global, delayed, evaluative, and synchronous. Prior knowledge of the effects of steering actions, as well as the natural dynamics (based on momentum and acceleration), are available as closed-form rules.
Pole Balancing	Sensors to detect a fallen pole	Feedback (signal indicating a fallen pole) is delayed, evaluative, structurally global, and asynchronous. Dynamics of the system can be made available as a reference in training.
Towers of Hanoi	Knowledge of a "desired state" and effects of various moves	Precise knowledge of the effect of each move on the external state is available a priori. Prior knowledge for partitioning the objective function is also available. Feedback is structurally global, delayed, evaluative, and synchronous.

learning problems in such environments, therefore, prefer learning the faster static strategies rather than the slower but more robust dynamic strategies.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* The feedback to the learning system depends on the formulation adopted. Analytical formulations based on combinatorial search are common for scheduling task graphs under zero natural dynamics for load; here, multiple alternatives can be evaluated and prescriptive feedback generated. In the case of empirical formulations, feedback is evaluative. Depending on the objective function, feedback may be available either after the completion of each job or after the completion of a batch of jobs.

The dependence of feedback on completion time introduces a delay between the occurrence of a decision and the arrival of feedback signal(s) evaluating it. In the interim, several other jobs may have arrived and been scheduled. This makes the feedback signal structurally and temporally global. Feedback is asynchronous because the time to complete a job (and generate the feedback) cannot be predicted.

Besides systems that use the scheduling formulation, load-balancing software is usually knowledge-lean. The effects of migrating jobs to remote sites cannot be predicted precisely. ■

Table 6 illustrates the concepts of this section on the strategy-learning problems described in the beginning of Section 2

2.4 Key Issues in Strategy Learning

Certain key issues are shared by a large number of strategy-learning problems, and their relevance depends on various characteristics of performance tasks, problem solvers, and learning environments. (For instance, prediction and TCA are especially relevant to tasks having time-varying problem variables.) Isolating such issues in discussing strategy learning methods will help us abstract problem-specific details, and allow us to consider those methods that are conventionally not used in strategy learning but are nevertheless appropriate for addressing these common issues. Table 7 summarizes these issues, characterizes the factors governing their relevance, and provides pointers to the pertinent approach(es). Table 8 summarizes the specific issues pertaining to some of the strategy-learning problems described in this section. The next section surveys approaches available for solving the various issues identified in this section.

3 Methods of Strategy Learning

Methods for strategy learning abound in the literature on machine learning, neural networks, cognitive science, and decision theory. While a comprehensive survey of these areas and methods can be attempted (see, for example, Mehra and Wah [75]), an *issue-based* survey is necessary for comparing and contrasting the well-known methods, as well as for identifying their common limitations. Table 7 summarizes the available approaches for addressing the issues raised in Section 2. In this section, we elaborate on these approaches, illustrating them using our running examples.

3.1 Tackling Ill-Posed Objectives

A strategy-learning problem is ill-posed when it lacks either a complete and accurate specification of its objective function (Issues 4, 5 and 6) or an appropriate way to combine multiple objectives (Issues 2 and 10). Consequently, it does not have a performance standard for choosing among the operators available at each decision point. The methods we describe in this section use past experience with problem solving to either model the objective function or learn other strategic information, which will help the problem solver compare the relative utilities of alternative moves.

Any of the methods described in this section can be adopted for learning while searching (Issue 6), provided that it addresses the tradeoff between exploration and convergence. While exploration demands that the system should sample a large number of states (usually by having a stochastic strategy), convergence demands that it should converge quickly to the true model of the objective function, and thence to the strategy that optimizes the function. A number of approaches for addressing this tradeoff rationally are now available in the literature [21, 76].

There are two general techniques for making a problem with an ill-defined objective well-posed: learning from absolute evaluation, and learning from relative evaluation. Both assume that the objective function is ill-defined and measurable over instants. Only the latter addresses the standard-of-comparison problem.

Table 7: Issues and approaches in strategy learning.

Issue (number)		Characteristics of Learning Task	Approach (where discussed)
Ill-posedness of objectives	Standard-of-comparison problem (4)	Ill-defined objective functions and evaluative feedback	Learning from relative evaluation (Sec. 3.1.2)
	Objective-function learning (5)	Ill-defined but measurable objective functions	Statistical regression when objective function is ill-defined, measured over instants; time-series regression, when measured over intervals (Sec. 3.1.1)
	Learning while searching (6)	Knowledge-lean learning environment, stochastic strategies, and asynchronous feedback	Learning models of reward-generation mechanisms and learning to predict improvements (Sec's 3.1.1, 3.1.2); must address trade-off between exploration and convergence
Credit assignment	Structural (13)	Structurally global feedback	Error-reducing approaches; problem-solver-specific implementations (Sec. 3.2.1)
	Temporal (14)	Temporally global (delayed) feedback	Complex general solutions, requiring reasoning with causal and temporal models; reduced complexity due to Markov property (Sec. 3.2.2)
Prediction	Predicting future states (1)	Time-varying problem variables and/or dynamic decision variables, especially difficult in knowledge-lean environments	Approaches based on projection and time-series analysis (Sec's 3.3.2, 3.3.3)
	Predicting future feedback (15)	Temporally global (delayed) evaluative feedback and/or ill-defined objective functions measurable over intervals	Modeling of the feedback-generation mechanism (Sec. 3.1.1) and approaches based on projection and time-series analysis (Sec. 3.3.1)
Violation of Markov Property (8)	Using past states and decisions in current decision	Non-Markovian representation of states in the problem solver, or lagged correlations in time-varying problem variables	Explicit storage of past decisions (Sec. 3.6) and explicit modeling of inter-decision relationships for temporal credit assignment (Sec. 3.2.2)
Constraint handling (7)		Exact well-defined constraints on solutions produced by strategies learned	Constraint satisfaction and constraint incorporation (Sec. 3.5)
Storing past decisions (9)		Non-Markovian representation of states in the problem solver, especially important with delayed feedback	Either on-line incorporation of feedback, or temporary storage of episodes (Sec. 3.6); methods for limiting the size of temporal scopes (Sec. 3.2.2)
Managing general objectives (10, 2)		Learning strategies for general problem solving; complexity of learning depends on the size of targeted problem class	Symbolic representation of problem spaces and general-purpose learning techniques (Sec. 3.7)
Dynamic decision making (3)		Dynamic strategies and dynamic decision variables; complexity increased by time-varying problem variables, violation of Markov property, and lack of prior knowledge of causal models	Dynamic programming and its variants used for Markovian representations (Sec. 3.4); no general solutions known for the non-Markovian case

Table 7 (cont'd) Issues and approaches in strategy learning.

Issue (number)		Characteristics of Learning Task	Approach (where discussed)
Handling structured solutions (11)		Indirect operators in the problem solver (usually with static strategies); especially complex in knowledge-lean environments	Preference for static strategies that consider multiple decisions at the same time; explicit representation of structured solutions during credit assignment (Sec's 3.2.1, 3.2.2)
Controlling nondeterminism (12)	Large search spaces for each instance prohibit search of multiple alternatives	The small number of operators in knowledge-lean environments and the generality of operator preconditions in knowledge-intensive environments; indirect nature of operators (thereby, the depth of solutions)	Preference for stochastic strategies that elegantly represent varying amounts of nondeterminism as randomness in search; learning while searching to limit episode size (Sec. 3.1.2)

3.1.1 Learning from Absolute Evaluation

This technique assumes that feedback $\mathcal{F}$ is directly related to the (unknown) objective function. It is useful for learning *evaluation functions* that can predict the true (or expected) objective value of states. We describe two methods: the first when feedback is immediate, prescriptive, and available for each state; the second, when it is immediate, evaluative, and possibly unavailable for some states.

a) *Learning the Objective-Function Value of States.* When states can be evaluated independently of each other, the path from the initial state to a chosen state can be optimized independent of the path from the chosen state to a goal state. One can induce an evaluation function using statistical regression techniques; such a function can then be used in place of a well-defined objective function [17, 15]. For example, in the game of checkers, the value of a board position is a function of the current board configuration irrespective of the moves made to reach that configuration. Therefore, several researchers [77, 28, 29] have designed systems that learn evaluation functions for this game.

When states can be evaluated only collectively, the evaluation of a state depends upon the path to it from the initial state. As a result, the objective function learned must be sensitive not only to the current state but also to some past states. The method of choice in this case is time-series regression [78, 79]. Two types of evaluation-function models are possible: i) those using a finite *window* of past states; and ii) those modeling evaluations as recurrences on state space.

Example 3 (cont'd). *Learning to Balance a Pole.* Lin [80] considers a version of the pole-balancing problem without the velocity inputs. This creates a non-Markovian state space. He considers both window-based and recurrent evaluation-function models, and finds that a window size of 1 works well. It makes sense because velocity can be estimated by applying the first-order differencing operator on distance. Lin also notes that recurrent models are to be preferred in general because it is difficult to guess the right window size. ■

Table 8: Examples of issues in strategy learning.

Example Task	Relevant Issues
Load Balancing (Example 1)	• "Minimum-completion-time" objective ill-defined, measurable over intervals [36]; • Standard-of-comparison problem due to lack of absolute evaluations [71]; • SCA over process-migration rules [64]; • TCA over interacting sequences of placement operations, complicated by a violation of the Markovian property [36]; • Prediction of future values of status variables [66]; • Capturing the effects of past states and decisions by learning abstract decision metrics (load averages) [36].
Steering a Ship (Example 2)	• SCA on preconditions of operators [13]; • TCA over solution structures comprising indirect operators [23]; • Prediction of trajectory under the influence of time-varying parameters as well as the controller's actions [72]; • General problem-solving and learning capabilities for diverse scenarios [32]; • Reduction of nondeterminism by macro-operator formation [73].
Pole Balancing (Example 3)	• "Balanced-pole" objective ill-defined but measurable over instants [49]; • Within-trial learning warranted by episode length and asynchronous timing of feedback [49] • TCA over sequences of balancing operations; history maintenance and eligibility computation simplified by the Markovian property [10]; • SCA over probabilities of applying "move-left" and "move-right" operations [27]; • Prediction of future feedback signals necessary for within-trial learning [70].
Towers of Hanoi (Example 4)	• General problem solving warranted by recursive transformation of the original goal into subgoals [31]; • SCA over heuristics for move selection [13]; • TCA over explicitly stored structured solutions due to indirect operators; • Explicit modeling of hard constraints [74]; • Controlling nondeterminism by macro-operator formation [73].

b) *Modeling the Reward-Generation Mechanism.* When external feedback is available only intermittently, an internal model — which characterizes feedback as a function of current and past values of problem variables — can be used for generating an internal feedback. Such a feedback signal can be used for evaluating the states not immediately followed by an external feedback signal [81, 9]. This approach works with evaluative feedback. However, in order to learn strategies from such feedback signals, the learner should know the relative importance of current and future signals, as well as whether feedback is to be minimized or maximized. Since learning from evaluative feedback is called reinforcement learning, internal generation of missing reinforcement is termed *secondary reinforcement learning* [9].

Example 3 (cont'd). *Learning to Balance a Pole.* Barto et al. [27] describe an approach using neural networks to learn a model of external feedback. Their approach performs learning while searching (Sec. 3.1). Besides learning to predict the 'pole-fallen' signal, it provides evaluations for states not evaluated by external feedback. Anderson [49] extends this work further using the more powerful multilayered networks of nonlinear sigmoidal units [82] to perform nonlinear regression. ■

3.1.2 Learning from Relative Evaluation

In environments having prescriptive feedback, the standard-of-comparison problem is relegated outside the learning system to the teacher. However, for evaluative feedback, it must be addressed by the learning system. The approaches described above do not address the standard-of-comparison problem [26] for learning from evaluative feedback in knowledge-lean environments (Issue 12). The methods we describe next can be used when neither the sign nor the absolute value of evaluation is important. Methods that learn from absolute evaluation can be thought of as *value-based*: they induce a model that can predict the objective-function value for any given state. The methods described below can then be thought of as *direction-based*: they estimate the 'slope' of the objective function. These methods assume only that the relative magnitude of evaluations is proportional to the relative goodness of solutions. They learn a restricted class of *dominance relations* [83, 2], which can be applied to either a state and its siblings, or a state and its ancestors, in order to select states for further exploration.

a) *Using Past States as Points of Reference.* Certain methods target improvement rather than optimization, and are useful when the degree of optimality of a solution cannot be computed easily. In this case, the current state is evaluated against either an incumbent (the best evaluation so far) or an average of recent values [84]. Thus, even though the original objective was ill-defined, the goal of learning is well-defined, namely, to improve upon the incumbent.

Example 3 (cont'd). *Learning to Balance a Pole.* An interesting approach to this problem is illustrated by the work of Morgan et al. [85] who use structurally local, immediate, evaluative feedback. They generate positive feedback for the part of the decision maker favoring 'move right' and negative feedback for the part favoring 'move left,' when the pole is to the right of vertical; and vice versa. The implicit goal of each part is to minimize feedback, which is generated when the pole either changes the side to which it leans, or changes significantly its angle of incline. When the feedback is received, the system uses only the immediately preceding state as a point of reference in order to improve the balance. ■

b) *Using Alternative States as Points of Reference.* For certain problems, especially those with objective functions measurable only over intervals, it is not clear how to define a point of reference using past states. However, if relative evaluations for alternative moves can be obtained, then the strategy learned should choose the move that has the best (relative) evaluation. Producing relative evaluations requires a mechanism that learns to compare consistently. For example, Tesauro [86] uses neural networks for learning to perform pairwise comparison of board configurations in learning strategies for the game of backgammon.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Completion times of jobs are important for determining the objective values for the load-balancing problem. Completion time is a function of both the time series generated by status variables and the current value of job-descriptor variables. The latter are unavailable for prediction, but their common effect can be ignored in comparing alternative operators that represent the migrations of the same job to alternative sites for execution. To perform a sensible comparison of alternatives, a reference state — characterized by no load on this site — is used as a common reference point. Regression or time-series regression, whichever is appropriate, is used for learning to compare the *relative* completion times of the same job under different loads [87]. ■

3.2 Solving the Credit-Assignment Problem

Exact and efficient algorithms for SCA (Issue 13) are available for a variety of decision structures; however, approaches to the TCA problem (Issues 11, 14, and 15) tend to be ad hoc and biased.

3.2.1 Solutions to Structural Credit Assignment

The eventual goal of SCA is to modify the decision processes so that the error between the actual and the expected outputs is reduced. Methods to achieve this goal need to consider the representations of solutions and strategies. Existing SCA algorithms adopt one of the following three representation techniques.

a) *Explicit Representation of Structured Solutions.* When the decision variables are dynamic and the problem solver's operators indirect, solutions are structured and consist of interdependent operator applications. This is particularly true of symbolic problem variables because decision rules are instantiated differently in different contexts. SCA for such tasks uses explicit representation of solutions, and distributes feedback backwards from the more recent decisions to the less recent ones. An example of such techniques is goal regression (pp. 300 of [88]), which is particularly suitable for learning from prescriptive feedback in knowledge-rich environments. In knowledge-lean environments, an approximation technique called experimental goal-regression [67] is more appropriate. Both techniques use the regression step, wherein new subgoals are derived from the unsatisfied preconditions of an operator, whose known postconditions match the current goal. If operator application leads to success, the preconditions of the operator are generalized either analytically (as in Explanation-Based Learning, or EBL [13, 89, 90]) or experimentally as in experimental goal-regression.

Example 5. *Learning to Solve Simultaneous Equations.* Porter [illegible] use experimental goal-regression to learn heuristics for solving simultaneous linear equations. The objective is to minimize the number of terms in the equations. The problem solver has operators that, for example, combine terms containing the same variable, combine constant terms, cancel zero-valued terms, substitute one equation into another, or replace an equation by its difference with another. The symbolic nature of terms and the indirect nature of operators give rise to structured solutions.

In their system, strategies are represented as preconditions for applying operators; these preconditions are initially set to some general test that always succeeds. Therefore, the problem solver initially finds solutions to given instances by search. When a solution is found, its specific facts become the preconditions for applying the operators involved. Parts of this solution structure may be found useful for solving another problem instance in the future; when that happens, the preconditions are generalized to accommodate both instances. The feedback signals merely indicate the applicability of operators, and SCA uses them to modify the existing strategy so that operators involved will apply correctly to a greater number of future instances. ■

A problem that complicates SCA with symbolic variables is the combinatorial complexity of the interactions among the variables to be considered. In this case, strategy learning can benefit from *factoring* the set of problem variables into groups of related variables; the simplified algorithm is called *independent credit assignment* (ICA) [88]. ICA is useful for learning from structurally local feedback.

b) *Explicit Representation of Decision Structures.* When solutions lack structure, a commonly used method for SCA is to represent explicitly, in richly connected networks, all possible ways in which decision-making rules can be combined. SCA in such structures consists of recording the level of activity for various components at the time of making a decision and then modifying either the components, their interconnections, or both.

Example 3 (cont'd). *Learning to Balance a Pole.* Anderson [49] uses a feed-forward neural network [82] to represent explicitly the decision structure of his pole-balancing system. Even though multiple decisions are made using the same network, all information used for credit assignment is stored as eligibility values with weights of the network. Weights are modified in the direction of more positive feedback, which constitutes a change of strategy. ■

c) *SCA with Decision Procedures.* Sometimes SCA is not possible because decisions are made in procedures, and even though the procedures may have modifiable parameters, the role of these parameters in decision making is not explicit. For example, in learning strategies for load balancing (Example 1), if the migration policy were implemented procedurally, and if the thresholds and weights used by it were available for modification, then some way other than traditional credit assignment must be found for altering the policy. One possibility is to model the effect of various parameters on (measured) performance. Using this model, SCA may be carried out as before. Another possibility is to have a population of candidate strategies that can be tested on different sets of instances. The best one can be selected based on measured average performance. Systems that simultaneously test multiple alternative strategies are said to be *population based* [91]. Population-based learning has been found to be a viable alternative to SCA, and its applicability is characterized mainly by the nature of its episodes. It has been used for learning new strategies for static load balancing of dependent jobs [92, 64], as well as for dynamic load balancing of independent jobs [93], and for designing suitable neural-network configurations [94, 95].

3.2.2 Solutions to Temporal Credit Assignment

TCA is responsible for disentangling the interrelationships between decisions and feedback. When solutions are structured, the decisions are causally related with each other. Explicit representation of such causal relationships can, therefore, simplify TCA; however, forming such representations requires complete and accurate knowledge of the consequences of applying each operator. In knowledge-rich learning environments, such knowledge is available to the learning system in the form of causal models. In knowledge-lean environments, there are two possibilities: the first is to heuristically determine the causal connections between decisions [96]; the second, to learn causal models in the course of problem solving [97]. While the first of these approaches introduces errors and biases into the learning process, the second incurs overhead during learning as well as problem solving. In general, the quality of knowledge-lean TCA procedures is somewhat suspect. Whenever possible, TCA should be avoided in knowledge-lean learning environments.

When problem variables vary with time, TCA requires not only a causal model but also a *temporal model* specifying the natural dynamics of the external state. Existing systems [98, 99] often make simplifying assumptions about these dynamics in order to simplify the model. For instance, Dean and Kanezawa [100, 24] classify operators into those that support the truth of a problem variable and those that do not. In their model, the probability of a Boolean variable being true increases exponentially towards one following a supporting action, and decays exponentially to zero following an interfering action. Exponential or memoryless dynamics are tantamount to assuming Markovian representations.

In general, TCA can be posed as a problem of determining the eligibility of a stored decision to receive a portion of the feedback signal [27]. Eligibility is computed using the causal and temporal relationships between the stored decision and the feedback signal. Thus, TCA amounts to resolving i) whether a decision could have caused the feedback, and ii) whether its effects were still persistent at the time of feedback generation [36]. Thus, TCA resolves the causal and temporal dependences between decisions and feedback. For example, in the pole-balancing problem, there is a causal dependence from a balancing decision to the state following it. There is also a temporal relationship, namely, the effect of a decision on a state decays exponentially with time. Typically, the causal component is used in an all-or-none fashion, to determine which decisions are at all eligible to receive feedback. On the other hand, the temporal model is used in the actual division of feedback because it provides a numerical eligibility value for each decision. Existing TCA algorithms follow three general schemes for representing and resolving decision interactions; these are described next.

a) *Explicit Representation of Causal Dependence.* Whenever a problem solver produces structured solutions, *i.e.*, a number of different operators contribute to the rewarded state(s) of the external environment, TCA can use the causal dependence to identify candidate decisions for receiving a share of the feedback signal. In certain cases [12], the depth of a decision in the causal chain, as well as its distance from the current state, are used for computing a numerical eligibility value.

Example 5 (cont'd). *Learning to Solve Simultaneous Equations.* A solution to a given set of equations comprises numerous operators applied one after another. Initially applicable operators, such as substitution and differencing, lead to states in which later operators, such as operators for combining variables and constants, can be applied. The causal dependences between earlier and later operators do not necessarily apply to all pairs of operators in the sequence. Structured representations, such as trees and graphs, are better at capturing the necessary causal relationships. More importantly, unlike the aforementioned sequence, these do not imply any unnecessary dependencies. The feedback for a solution can be distributed either equally to all decisions that are causally connected to it [11], or proportionately based on the distance from the decision to the final state [67]. ∎

b) *Explicit Representation of Decision Structures.* In systems with common algorithms for SCA and TCA [101, 102], learning can be simplified if the solutions do not have an internal structure. As noted above, such systems represent their decision structure explicitly, which includes all plausible interactions among decision rules. The algorithms proposed for computing the eligibilities of parameters in these rules are variants of Sutton's *Time Difference* (TD) procedure [70], which have been shown to be rational for problems satisfying the Markovian property.

Example 3 (cont'd). *Learning to Balance a Pole.* Barto et al. [27] use a simple neural network as the $\mathcal{PS}$ component of their learning system for pole balancing. Feedforward neural networks are explicit decision structures. (Certain other control architectures [103] employ recurrent networks, which cannot be considered explicit decision structures because the full context of decision making is not stored in such a structure.) Because of the Markovian nature of the representation used by the $\mathcal{PS}$, the temporal dependence between decisions and feedback follows an exponentially decaying pattern. Sutton [70] has shown that, for Markovian representations, eligibility can be updated incrementally. Since no additional information from past states and/or decisions is required, the eligibility values stored with each weight contain sufficient information for TCA. ■

c) Scoping of Decisions. This approach is characterized by a dynamic history of past decisions [36] and is especially suitable for problems having either multiple or structured objectives and time-varying problem variables. Like the first approach above, it keeps an explicit record of decision making; and like the second, it computes numerical eligibility values. However, instead of associating eligibility values with decision parameters, it associates them with decisions themselves. The essential information with each decision is its scope: if a feedback signal falls within a decision's scope, then this decision is eligible to receive a portion of the signal. Two types of scope information are used.

Causal scope can be used for separating decisions pertinent to each component (when several components of an objective function can be computed independently) or each objective (when multiple objectives are evaluated by the same feedback signal). Causal scoping can prevent decisions from receiving a share of the feedback caused purely by the natural dynamics of the external state.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* The decision to migrate a job has the effect of reducing the load at the originating site and of increasing the load at the destination site. The causal scope of a migration decision includes only those other decisions that made use of information about the load at either the source or the destination site. When a job finishes, its completion time is evaluated against the case without load balancing, and an evaluation of the scheduling decisions generated based on such evaluation. A decision is eligible to receive a share of the feedback only when its causal scope includes placement or migration decisions regarding the completed job. ■

Temporal scope (or extent). A state or a decision is said to be *in the temporal extent* of another (earlier) decision if the effects of the latter persist until the occurrence of the former. When the external state of the environment is fully controlled by the problem solver, decisions can have an infinite extent. More often though, decisions have a finite temporal scope and can be discarded from the history of past decisions without affecting learning. In addition, the learner can model the persistence of each decision (the degree to which a decision affects a feedback signal that occurs within the decision's extent). Such a temporal model may take several forms. When objective functions evaluate states independently (usually for Markovian representations), and when decisions affect the external state immediately, a temporal model specifying an exponentially decaying persistence is quite adequate. In more complex environments, where objective functions are evaluated over intervals, or where decisions take effect after an initial delay, more complicated temporal models may be needed.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Migration of a job causes load changes at both the site of origin and the destination site. The effects are felt while the job is executing; after a job finishes execution, future decisions are not affected by the migration decision in question. In this sense, migration decisions have *finite temporal scopes*. Within its temporal scope, a decision's persistence initially increases while the job is being migrated, then stays high during the job's lifetime, and drops quickly to zero again upon completion. When a migration decision is determined using causal scope to be a candidate for receiving feedback, it is given a portion of the feedback signal proportional to its persistence at the time of feedback. ■

3.3 Prediction

Prediction problems are inherent to performance tasks having time-varying inputs (Issues 1 and 15). A number of approaches, each with its own advantages and disadvantages, are available in the literature. Various factors affecting the choice between these include i) on-line versus off-line learning, ii) numeric versus symbolic problem variables, and iii) dynamic versus static decision variables.

3.3.1 Temporal Difference Methods

Temporal difference (TD) methods originated in early literature on dynamic programming and problem solving, and methods related to learning were studied formally by Sutton [70]. The essential idea of these on-line methods is to rewrite the total error of prediction as a sum of differences between successive predictions. If the problem solver uses Markovian representations, numeric problem variables, and direct objectives, then the error of prediction can be minimized by reducing the error between successive predictions [69, 30].

Example 3 (cont'd). *Learning to Balance a Pole.* A popular class of strategy-learning algorithms, known as ACE/ASE, [27, 104] use TD methods to predict the discounted sum of all future evaluations at each time step while simultaneously using the difference between successive predictions to modify the predictor. (In the computation of this 'discounted' sum [68], immediate evaluations are given exponentially greater weight than their successors.) The predicted sum of future evaluations can be used instead of missing external evaluations as the internal feedback, thereby solving the TCA problem. The error of prediction is divided between prediction decisions using a recency weighting [70]: older predictions incur exponentially smaller errors than recent predictions when computing the current difference between successive predictions. This is a justifiable heuristic for Markovian representations and instant-evaluated feedback, both of which are true of the pole-balancing problem. ■

3.3.2 Time-Series Analysis

This refers a large class of approaches that build models using sequential periodic (therefore, synchronous) observations of time-varying parameters, and predict based on these models. Time-series methods have been developed by researchers in statistics, neural networks, and control. In general, these are useful for predicting future states of learning problems having numeric problem variables and static decision variables.

Time-series models developed by statisticians [105, 25] comprise i) trends: linear, polynomial, and exponential dependence on time; ii) periodic components: as found by frequency-domain analysis of a signal; and iii) random or unmodeled dynamics of environmental processes. Statistical time-series analysis techniques tend to be off-line and require more complex computations than on-line ones, such as the TD method described above.

Nonlinear recurrent neural networks [106] can be used for time-series prediction. However, they have not been popular in strategy-learning systems due to the unresolved problems of instability, poor convergence, and chaotic behavior. Somewhat more success has been obtained with TDNNs (time-delay neural networks) [107], which are non-recurrent but simultaneously examine several delayed values of problem variables along with their current values. Both recurrent and non-recurrent neural networks are capable of on-line learning.

While the methods described above are applicable to prediction in the absence of prior knowledge, problems with numeric time-varying parameters, Markovian representation, dynamic decision variables, and knowledge-rich environment can be solved using Kalman filtering [108]. Kalman filtering is an example of state-space methods developed by control theorists [109]. Its predictions as well as its model of environmental dynamics can be updated on-line in straightforward steps.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* The status variables in load balancing, namely, the levels of utilization on various resources, vary with time and often exhibit periodic (stationary with time) behavior as well as local trends (non-stationary behavior). These characteristics are due to the fact that the background load is generated by a process population whose rate of change is slower than the sampling rate of status variables. Statistical time-series methods such as ARIMA modeling [37] are useful for identifying trends in these series, which can then be used in making short-term predictions of future load. ■

3.3.3 Projection

Methods of temporal projection were developed as part of temporal-reasoning systems in AI. These employ a technique called forward chaining: the current state of the world and a set of rules that fire therein generate new facts that characterize a possible next state of the world. Using these new worlds, one can continue to reason and to get a two-step-ahead prediction, and so on [100, 110]. Heuristics of preference [24, 98, 99] are needed for narrowing down the space of possible worlds. Time, being a continuous quantity, does not easily lend itself to modeling with discrete event-based symbolic representations except when problem variables do not change outside the control of the problem solver. Projection methods are, therefore, applicable to symbolic (but not time-varying) problem variables and dynamic decision variables. Dean and Siegle [111] have proposed approaches that can reason symbolically about natural dynamics specified using differential equations.

Strategy-learning systems for symbolic problem variables and dynamic decision variables in time-varying environments commonly use reactive rather than proactive approaches. Such methods are known as reactive planning [53, 55, 54, 56] and learning from failure [112, 113, 49]. In some sense, these approaches ignore the natural dynamics of environmental processes and predict (using chaining) only the causal dynamics of the process.

Example 2 (cont'd). *Learning to Steer a Ship.* Although the ship-steering problem involves decision-making with time-varying problem variables, there is little time for the problem solver to search in the space of solutions. Instead, the problem solver uses a causal model that relates its control inputs and resultant states. The model is ***learned from observation*** by relating patterns in recent control actions with patterns (trajectories) in state variables following these actions. It is then used in chaining to determine action sequences that will reach a desired state from the current state. ■

3.4 Exploiting the Markovian Property

Markovian representations often accompany objective functions that are either well-defined or ill-defined and measured over instants (Issue 3). Some of these well-defined functions satisfy the path-independence axiom; for these, in the presence of deterministic strategies, immediate feedback, and complete knowledge of state transitions (knowledge-rich environment), the optimal solution can be computed using dynamic programming [42, 114]. Similarly, for ill-defined functions, the objective of maximizing the (possibly discounted) sum of future evaluations is also amenable to dynamic programming [68]. Variants of this procedure exist for knowledge-lean environments as well as for problem solvers with stochastic strategies [115]. In this section, we discuss three different dynamic-programming formulations for strategy learning. Note that these formulations are limited by their requirement of a finite number of states and their high complexity in the presence of a large state space.

3.4.1 Dynamic Programming (DP)

Approaches based on pure DP are suitable for knowledge-rich environments, symbolic or discrete problem variables, and Markovian representations. When the assumptions of DP are satisfied, the optimal strategy can be computed given the objective function (defined either as utility or as cost) and a causal model of the environment. However, without additional heuristics to restrict the space of possible solutions, DP is too complex to be practical. One way to harness this complexity is by learning dominance relations between states [116], which can be used to prune moves leading to suboptimal states.

Example 4 (cont'd). *Learning Strategies for the Towers of Hanoi Problem.* Dynamic programming can be applied when the objective is to find a solution that transforms the initial state to the goal state in the minimum number of moves. Since operators applied result in deterministic changes to the problem state, DP can be used. (For probabilistic transitions, the SDP algorithm described next can be used.) The first step is to set up the functional equations of dynamic programming [117], which define recursively the optimal cost of the final state(s) in terms of the optimal costs of the preceding states. In the Towers of Hanoi problem, there are two possible preceding states: either the smallest disk was moved from the left-most tower or the middle tower. The optimum cost of each of these states can again be computed using similar recursive equations. Depending on which preceding state achieves the minimum cost in the recurrence, the optimal solution can be constructed. The major drawback of dynamic programming is that it may require computations exponential in the number of problem variables. ■

Dynamic programming can aid strategy learning by generating complete optimal solutions to certain specific instances; one can then use credit-assignment procedures [8] or generalization techniques [72] to generalize these solutions to new goal states as well as new instances.

3.4.2 Stochastic Dynamic Programming (SDP)

When a problem solver's strategy is stochastic, so is the evaluation of the solution. In this case, it is not possible to say whether one policy is better than another for a given instance; instead, one needs to consider the expected evaluation of the policy rather than its exact evaluation [14, 118]. Methods analogous to deterministic DP can be used to compute the optimal strategy. Stochastic dominance relations between states [2] compare their expected evaluations rather than the exact ones, and help limit the complexity of SDP. Their use leads directly to a method for refining strategies called *policy iteration* [68]. However, like DP, SDP also requires extensive knowledge of the environment and a well-defined objective function.

Example 6. *Learning the Optimal Route to a Goal around Barriers on a Grid.* This is the route-finding problem discussed by Barto et al. [14] in their review of exact and approximate methods for SDP. The performance task has an ill-defined but measurable objective of reaching a target location on a two-dimensional grid, starting from a given location on the grid. It also has an ill-defined constraint that the path should not cross certain initially unknown barrier locations. The environment produces an immediate feedback signal whose value is -1 for all states except the goal state. The problem solver has four operators (whose preconditions are unknown but effects are known): one each for moving up, down, left, and right. The problem solver's state includes information about the current location on the grid. The state space is Markovian; and the operators, direct. The problem solver's strategy is stochastic, associating a probability of applying an operator with each state. Equivalently, the strategy defines a Markov chain on the state space, where transitions are possible from each state to one of its four neighbors. No transitions are possible out of the goal state.

With each strategy, SDP associates an evaluation function that allows one to compute, for each state, the expected number of steps to the target using the current strategy. As in deterministic DP, one can translate the optimality of strategies into the optimality of their respective evaluation functions [118]. SDP also permits one to set up a recurrence equation for computing the optimal path length with a k-step lookahead in terms of the optimal length for $(k-1)$-step lookahead. Once the optimal evaluation function is computed by solving this recurrence, the optimal strategy simply picks the action that optimizes this function at each step. ■

3.4.3 Heuristic Dynamic Programming (HDP)

HDP [119, 115, 21] is applicable in knowledge-lean environments where a complete model of state transitions is unavailable. As discussed before, DP derives an optimal strategy by first deriving an optimal evaluation function from a partial enumeration of the search space. HDP, on the other hand, works for problems with ill-defined objective functions by estimating an optimal evaluation function and an optimal strategy. In the absence of extensive knowledge of state transitions, the learned evaluation function and strategy can only approximate the optimal ones.

Example 3 (cont'd). *Learning to Balance a Pole.* The basic idea of HDP is to use a generalization of temporal difference methods to predict the sum of all future reinforcements. At any point in time, the prediction process is adapted so it will predict correctly the sum of the next state's prediction (using the current predictor), the external feedback received in that state, and an appropriate negative constant (to keep the sum of future reinforcements finite) [119]. The prediction at the next state using the current set of weights becomes the apportioned feedback for the current decision. The necessary strategy modifications for SCA can then be carried out using this feedback. ■

3.5 Constraint Handling

Problem solvers take two distinct approaches to constraints: i) by satisfying them explicitly, and ii) by incorporating them into the solution procedures (Issue 7). Accordingly, strategy-learning systems can modify either the operator-selection procedure or the objective function in order to prefer moves that generate valid states rather than those that generate invalid ones. A common solution is to induce a model of the feasible region from empirical observation of feasibility and infeasibility of various states. The model (like soft constraints) may then be incorporated into the objective function as penalty terms. Soft constraints can often be incorporated as (positive) penalty terms added to the objective function of a (minimization) problem [120]. Unknown or ill-defined soft constraints are handled just as unknown or ill-defined objectives. In contrast, hard constraints must be modeled explicitly [74] so that the learning system does not over-generalize from limited experience. Techniques such as Lagrange multipliers [120] are useful for incorporating hard constraints into objective functions during problem solving. However, they increase the effective problem size and, therefore, the complexity of the problem.

3.5.1 Constraint Satisfaction

Constraint-satisfaction procedures for learning with symbolic (discrete) problem variables are typified by explanation-based learning (EBL) methods in which generalization is performed by retaining the structure of a known solution and by relaxing the conditions under which the structure applies. In this case, constraint satisfaction can reduce the space of solutions significantly. General techniques for finding feasible solutions for constrained problems with discrete-valued problem variables are surveyed by Nadel [121]. Techniques based on truth maintenance can be used to enforce symbolic constraints during decision making and learning. (See reference [122] for an overview.)

Constraint-satisfaction approaches for learning with continuous variables induce a model of the feasible region and use it to perform constrained optimization during both decision-making and learning. Examples include explicit search for feasible solutions, and subsequent training of decision makers using such solutions as examples [123]. Methods for constrained optimization of continuous functions are reviewed by Walsh [120] and use projection of infeasible solutions to the nearest point in the feasible region. Such methods are applicable only with well-defined constraints, but allow great flexibility for the strategy learner.

Example 4 (cont'd). *Learning Strategies for the Towers of Hanoi Problem.* The constraint, that if a disk A is on top of another disk B then A must be smaller than B, is part of the domain theory. New states are generated by applying operators; those that violate this constraint are simply inconsistent with the domain theory. Upon detecting that the state resulting from the application of an operator meets all other constraints except this one, the problem solver may postpone applying this operator until the constraint can be satisfied. In the meanwhile, the problem solver may attempt to satisfy this constraint by setting up alternative subgoals [72]. ■

3.5.2 Constraint Incorporation

When the problem variables are numeric, constraints can be incorporated into the objective functions as penalty terms. Jordan [103] describes one approach for incorporating constraints into the SCA process and another for incorporating constraints into the cost function. His approaches work for a variety of numerical constraints in both knowledge-rich (constraints are explicit) and knowledge-lean (constraints are available implicitly by random sampling) environments. Similar approaches are discussed under the rubric of knowledge compilation. An example is the test-incorporation approach of Dietterich and Bennett [124]. The drawback of methods in this class is that different relative importance given to the objective function and constraint terms leads to different optimal solutions.

Example 6 (cont'd). *Learning the Optimal Route to a Goal around Barriers on a Grid.* In this case, the constraints are ill-defined in the sense that their violation can be detected only when the problem solver attempts to move into the barrier.

The net effect of constraint violation is the increased cost of solutions because infeasible moves cause no change in state and are, therefore, wasted. Thus, by optimizing the cost function, the problem solver learns to avoid barriers automatically. ■

3.6 Managing a History of Recent Decisions

Available solutions to this problem use either the episode structure of problem instances [11, 13] or Markov property [49, 125] to limit the amount of information stored (Issues 8 and 9). When the problem variables do not vary with time, the state of the external environment does not change outside the control of the problem solver; in this case, the episode ends when the final feedback signal related to this instance is received. This feedback is used either for modifying the decision process via credit assignment, or for assessing the importance of the current solution path. In some cases, entire solution paths, or generalizations thereof, are retained for solving similar problem instances in the future. Examples include analogical learning systems [126, 127], that retrieve and then deductively transform old solutions for solving new problem instances.

When the problem variables vary with time but their distributions are stationary, a learning system may still be able to converge to an optimal control strategy in either an absolute sense (for deterministic variation) or an average sense (for stochastic variation). One may consider such convergence as the end of an episode. Frequently though, either the length of episodes or the non-stationarity in the environment precludes storage of complete solutions. When Markov property is violated, new states and decisions must be added or removed continuously from the history. A simple solution is to keep a fixed window of past states and decisions; see, for example, the discussion of Lin's pole-balancing approach in Sec. 3.1.1a.

Sutton [70] proposes the use of a scalar 'eligibility' value for each modifiable parameter of the decision maker. Holland [128] maintains a 'strength' parameter with each rule, which represents the average credit received by that rule over several problem instances. Thus, scalar indicators of eligibility can be associated with either individual parameters or rules. Yet another option is to associate such indicators with each decision stored in the history [75]. When past state information is the trace of a time-varying problem parameter, time-series methods similar to those used for learning objective functions may be employed for automatically constructing abstract problem variables, which can then be used in decision making [44]. The naive alternative — to maintain the full state vector and associated decision variables and to process feedback signals using prior knowledge — is both expensive and unsuitable for knowledge-lean environments.

Example 1 (cont'd). *Learning Strategies for Load Balancing.* Load-balancing decisions have finite temporal scopes (Section 2.3.1). The non-Markovian learning problem requires decisions to be kept until feedback is received. Decisions with expired temporal scope are not eligible for feedback and can be deleted [44]. ■

3.7 General Problem Solving and Learning

Systems for general problem solving requires symbolic-reasoning capabilities (Issues 10 and 2), irrespective of the nature of problem variables. Such systems are capable of general strategy learning, although extensive amounts of explicitly stated task-specific knowledge are needed. The techniques used by such systems include knowledge-compilation methods, such as macro-operator formation [72], chunking [129, 130], and procedure learning [131]. The archetypical system in this class is SOAR [132] which can learn strategies associated with arbitrary problem spaces defined by objectives and constraints using symbolic variables.

As for generalization to other instances of the same performance task variables that occurs spontaneously, those tasks that have symbolic problem variables must generalize explicitly, using algorithms such as AQ15 [133] and the version-space method [134]. The AQ15 algorithm is more powerful of these two because it allows instance descriptions to have both symbolic and numeric components. Specifically, these algorithms can be used for generalizing preconditions of operators upon success and specializing them upon failure.

Example 7. *SAGE.2: A General Learning System.* Langley [8] demonstrates his SAGE.2 system on the Towers of Hanoi (Example 4) and five other learning problems. This program employs general-purpose learning rules that allow it to characterize good solutions in many domains. For instance, one such rule takes a complete solution tree and characterizes the states on the path from the initial state to the goal state as good, while characterizing all other states as bad. Yet another general learning technique specializes preconditions of operator-application rules so that a rule will fire only when its firing does not result in a bad state. The same general rule is shown useful in learning strategies for several diverse problems. ■

The techniques described in this section resolve most of the issues raised in Section 3. Tables 7 and 8 together describe how to handle the issues relevant to our running examples. Consider, for instance, the load-balancing problem. Knowing that its objective function is ill-defined and measurable over intervals and that absolute evaluations are unavailable, we should apply time-series regression using an alternative state as a point of reference in order to address the ill-defined nature of its objective function. For the ship-steering problem, which requires symbolic prediction of the ship's trajectory, the temporal-projection methods of AI are the appropriate technique.

To summarize, we have presented in Sections 2 and 3 a general method for identifying the issues and techniques relevant to any given strategy-learning problem. We first characterize the problem using various attributes of its performance task, problem solver, and learning environment. This allows us to focus our attention on issues relevant to learning strategies for that problem type. We then select appropriate methods for addressing those issues. Next, we examine different ways of building an operational learning system out of the building blocks described above.

4 Architectures for Strategy Learning

Figure 2 shows the basic information flow of a strategy-learning system. The type of strategy-learning problem(s) targeted determines the method used for implementing the strategy-learning system. In this section, we show four different architectures for strategy learning, each motivated by a different class of learning problems. The first three models considered here are *point-based* in the sense that they keep track of one point at a time in the space of possible strategies; the final model considered here is *population-based* in that it simultaneously keeps track of multiple strategies.

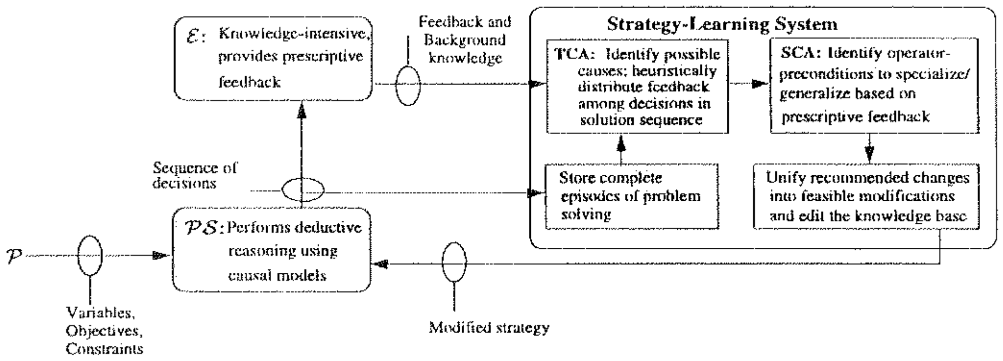
4.1 Knowledge-Based Model

Learning systems developed in the areas of cognitive science and artificial intelligence tend to have knowledge-rich learning environments and synchronous, delayed, prescriptive feedback. The structure and operation of many of these learning systems can be described, as in Figure 6a, using the knowledge-based model of strategy learning. This models fits well the models described by Dietterich and Buchanan [1, 23], Langley [8], and Smith *et al.* [135].

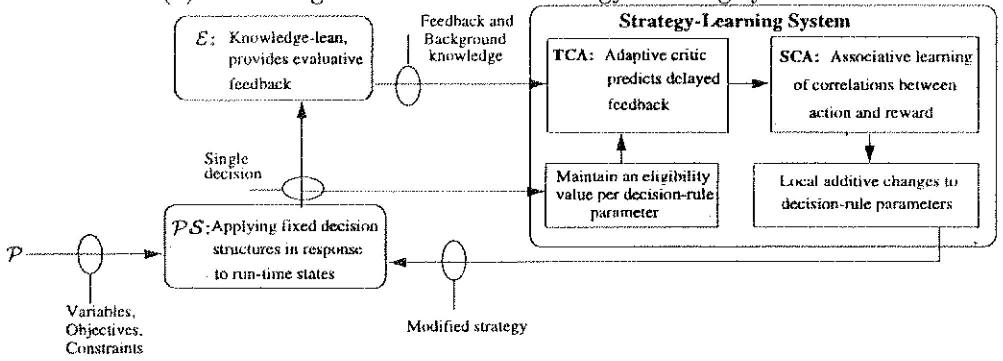
The performance tasks of interest are characterized by i) symbolic problem variables whose values do not vary with time, ii) well-defined objective functions and general problem solving, and iii) well-defined hard constraints. The problem solvers usually employ rule-based deductive techniques using static deterministic strategies, indirect operators, and non-Markovian representations. The learning environment is information-rich where background knowledge of the problem domain abounds and the learning system attempts to exploit it by learning a lot from each experiment. This knowledge often takes the form of causal models relating preconditions and postconditions of operators.

Since objectives are well-defined, none of the issues caused by ill-posedness of objectives (Table 7) are relevant here. The problem solvers handle dynamic decision making and prediction of future states by deductively reasoning with the causal models. Because problem variables do not vary with time and because of the availability of background knowledge, the problem solvers employ static strategies that generate entire solution sequences at once. Structured solutions caused by indirect operators are explicitly represented, upon which SCA is performed by regressing the explicitly defined objectives through the causal model of operators. Temporal credit assignment is performed by explicitly representing causal chains linking states and decisions. History management is done by storing complete episodes. Constraints are handled using constraint-satisfaction approaches.

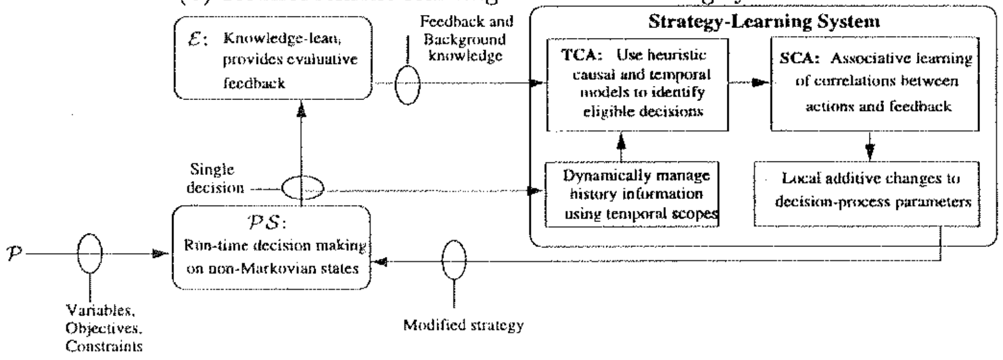
The credit-assignment modules make extensive use of stored knowledge both to relate the external feedback to objectives and to determine the preconditions of operators responsible for the current feedback. The learning system uses an internal model of the problem solver to edit strategies. Retention of complete solutions permits the use of symbolic learning techniques, such as syntactic generalization of stored solutions and formation of macro-operators. Syntactic generalization and credit assignment based on abductive reasoning permit systems based on this model to generalize substantially from each example. Rule schemata, which can be derived using only a few episodes of successful problem solving, capture the common struc-


$\mathcal{E}$: Knowledge-intensive, provides prescriptive feedback
Feedback and Background knowledge
Strategy-Learning System
TCA: Identify possible causes; heuristically distribute feedback among decisions in solution sequence
SCA: Identify operator-preconditions to specialize/generalize based on prescriptive feedback
Sequence of decisions
Store complete episodes of problem solving
Unify recommended changes into feasible modifications and edit the knowledge base
$\mathcal{P}$
$\mathcal{PS}$: Performs deductive reasoning using causal models
Variables, Objectives, Constraints
Modified strategy

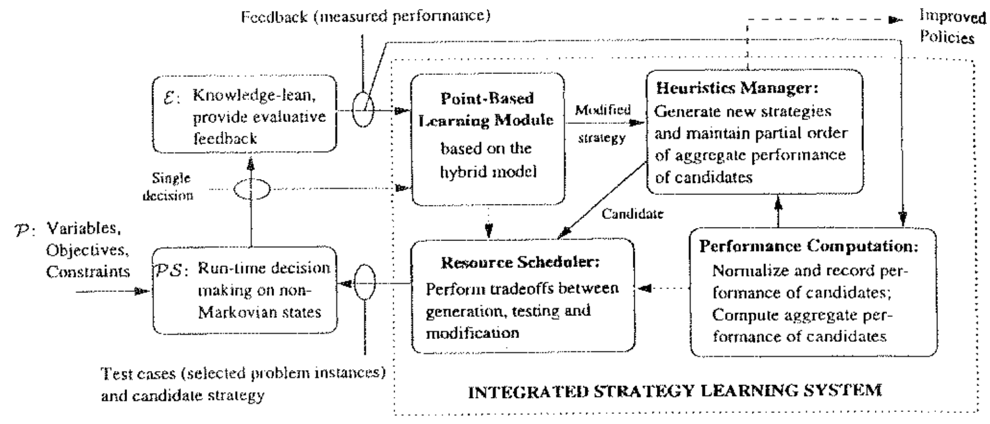
(a) Knowledge-based model of strategy-learning systems


$\mathcal{E}$: Knowledge-lean, provides evaluative feedback
Feedback and Background knowledge
Strategy-Learning System
TCA: Adaptive critic predicts delayed feedback
SCA: Associative learning of correlations between action and reward
Single decision
Maintain an eligibility value per decision-rule parameter
Local additive changes to decision-rule parameters
$\mathcal{P}$
$\mathcal{PS}$: Applying fixed decision structures in response to run-time states
Variables, Objectives, Constraints
Modified strategy

(b) Reinforcement-learning model of learning systems


$\mathcal{E}$: Knowledge-lean, provides evaluative feedback
Feedback and Background knowledge
Strategy-Learning System
TCA: Use heuristic causal and temporal models to identify eligible decisions
SCA: Associative learning of correlations between actions and feedback
Single decision
Dynamically manage history information using temporal scopes
Local additive changes to decision-process parameters
$\mathcal{P}$
$\mathcal{PS}$: Run-time decision making on non-Markovian states
Variables, Objectives, Constraints
Modified strategy

(c) The hybrid point-based model of learning systems


Feedback (measured performance)
Improved Policies
$\mathcal{E}$: Knowledge-lean, provide evaluative feedback
Point-Based Learning Module based on the hybrid model
Modified strategy
Heuristics Manager: Generate new strategies and maintain partial order of aggregate performance of candidates
Single decision
Candidate
$\mathcal{P}$: Variables, Objectives, Constraints
$\mathcal{PS}$: Run-time decision making on non-Markovian states
Resource Scheduler: Perform tradeoffs between generation, testing and modification
Performance Computation: Normalize and record performance of candidates; Compute aggregate performance of candidates
Test cases (selected problem instances) and candidate strategy
INTEGRATED STRATEGY LEARNING SYSTEM

(d) Population-based model of learning systems

Figure 6: The four strategy learning architectures.

ture of observed solutions; their preconditions are the maximally general conditions under which the operators of the common structure may be applied.

4.2 Reinforcement-Learning Model

Learning systems in the areas of control and decision theory are expected to operate in knowledge-lean learning environments. Techniques developed in these areas generally employ statistical methods to identify models, and estimate their missing parameters. Knowledge-lean environments usually produce evaluative feedback which, in the presence of time-varying problem variables, makes TCA the principal problem. The common characteristics of systems employing statistical methods for learning are well represented by the reinforcement-learning model of strategy learning (Figure 6b). This model underlies the AHC/ASE models of Barto et al. [27] as well as the reinforcement-learning model of Minsky [9].

The key characteristic of the strategy-learning problems of interest here is that the representations employed by the problem solver satisfy the Markov property and states can be evaluated independently of each other. The performance tasks of interest are characterized by i) numeric, time-varying problem variables, ii) dynamic decision variables, iii) ill-defined and specific objectives, and iv) soft constraints. The problem solvers employ stochastic, dynamic strategies, direct operators, and Markovian representations. The learning environment is knowledge-lean and feedback is delayed, evaluative, and asynchronous.

The objective-function learning problem is solved by learning evaluation functions of states; the standard-of-comparison problem, by comparing future states against the current state; and the problem of learning while searching, by the use of stochastic strategies whose randomness is reduced as learning progresses. The problem solvers perform dynamic decision making by exploiting Markov property to pick (with high probability) the maximum-utility move at each decision point. The soft constraints are handled using constraint-incorporation. Temporal credit assignment in this model is characterized by its use of feedback predictors (also known as *adaptive critics* or *secondary reinforcement devices*), which learn to produce internal feedback whenever the external feedback is delayed. Such prediction of future feedback often employs the TD methods of Sutton [70]. Structural credit assignment is performed on static decision structures (typically maintained using feedforward neural networks [136, 49]) using the back-propagation algorithm [82] and its variants. Since Markov property holds, history-management issues are not relevant here.

Statistical methods are useful as ways to bootstrap learning without prior knowledge, but such (associative) learning techniques require a large number of problem-solving episodes. The amount of information extracted from each example is small relative to the knowledge-based model. That causes slowness of learning, which is an important factor behind several recent proposals for hybrid reinforcement-learning architectures [125, 34].

4.3 Hybrid Point-Based Learning Model

The hybrid point-based learning model [75, 36, 44] (Figure 6c) was motivated by non-Markovian strategy-learning problems in knowledge-lean environments. The standard reinforcement-learning model is not directly applicable to such problems because it lacks both the knowledge and the reasoning mechanisms necessary for distributing credit among explicitly stored past decisions [30, 80]. Usually, knowledge-lean environments are accompanied by ill-defined objectives and evaluative feedback, both of which preclude the use of the knowledge-based model.

In the hybrid model, the states and decisions are recorded in a dynamically managed history as decisions are made. Temporal models, containing information about persistence and temporal scopes, are used in history management as well as in TCA. Causal models are used only to identify candidate decisions during TCA. Relative to the knowledge-based model, the causal model used here can be heuristic and less detailed. Unlike the standard reinforcement-learning model, the hybrid model cannot integrate solutions for TCA and SCA into the same algorithm: instead of associating eligibility values with modifiable parameters of the problem-solver's strategies, it associates eligibility values with individual decisions in the history.

When feedback becomes available, the candidate decisions are first identified using the heuristic causal model; the eligibilities of individual decisions are then used for proportional assignment of credit/blame among decisions. SCA is performed on the explicitly stored portions of solutions. Examples of systems employing the hybrid model include Samuel's Checker Player [29], Widrow, *et al.*'s truck-backer-upper system [65], and learning systems based on genetic algorithms [137, 138].

4.4 Population-Based Learning Model

All the models described above employ point-based search in the space of strategies, using credit assignment to modify the incumbent strategy based on feedback, and either stochastic strategies or explicit perturbations to explore the search space. The fundamental idea in population-based learning [91] (Figure 6d) is to use population-based methods for probing the strategy space at several points simultaneously. Starting with an initial pool of candidates, this model admits several ways of generating new candidates: i) *grammar-based*, in which problem-solving strategies are generated as leaves of a phrase-structured grammar; ii) *perturbation-based*, in which random or systematic perturbations are applied to the incumbent strategy to obtain new candidate strategies; and iii) *performance-based*, in which credit assignment is used either for modifying the incumbent or for updating the probability of selecting one of the candidates generated.

The population-based model is characterized by its resource scheduler, which rationally divides learning time between generation, testing, and modification. It honors deadlines on learning time. In order to make rational and effective use of the limited time, it uses the theory of sequential selection from statistical decision theory [139]. However, this theory applies under certain restrictive assumptions on the distributions of objective-function values over the populations of problem instances.

Population-based learning works well when episodes are short, experimentation is inexpensive, and the environment too complex to model. This model targets breeding and selection of strategies instead of modification. It, therefore, avoids, credit-assignment problems. The standard-of-comparison problem is solved by either comparing strategies on an instance-by-instance basis or by normalizing the performance of all strategies with respect to the average performance of a baseline strategy. Objective-function learning is achieved via a statistical sampling process that seeks to estimate the mean and variance of a strategy's performance. The degree of confidence sought is just enough to allow the learning system to minimize the risk of choosing a bad strategy due to insufficient testing.

Population-based learning has been used to learn strategies for static load balancing with dependent jobs [92], dynamic load balancing with independent jobs [38], VLSI test generation [140], and stereo vision [141].

Table 9 summarizes the characteristic features of the four architectural models. Table 10 reviews their applicability to the strategy-learning problems described at the end of Section 1.

5 Conclusions

A strategy-learning problem is a triple comprising a performance task, a problem solver, and a learning environment. Performance tasks are characterized by their variables, objectives, and constraints; problem solvers, by their representations, operators, and strategies; and learning environments, by their feedback and knowledge-intensity. Strategy-learning problems drawn from diverse fields can be classified using this taxonomy.

The issues of interest to the designers of strategy-learning systems depend on various attributes of the learning problem(s) they wish to solve. Complex learning problems are characterized by ill-posed objective functions, delayed feedback, violation of Markov property, dynamic decision making, time-varying problem variables,

Table 9: Architectural models characterized by their approach.

Model	Issue	Approach
K	Credit Assignment	Knowledge-based algorithms using ICA (Sec. 3.2.1); explicitly represented structured solutions; well-defined objective functions
	Prediction	Projection of symbolic variables using explicit causal model
	General Problem Solving, Non-Markovian Representations, Indirect Operators, Multiple General Objectives, Storing Past Decisions	Using powerful symbolic representations, prior knowledge of general purpose problem-solving techniques, static strategies in problem solvers, and explicit storage of structured solutions
R	Ill-Posed Objective Functions	Interleaved learning of instant-evaluated objective functions and dynamic strategies; learning from absolute evaluation and (delayed) evaluative feedback
	Credit Assignment	Complex TCA problems in knowledge-lean environments solved using on-line learning of reward-generation mechanism; SCA interleaved with TCA; not suitable for non-Markovian representations
	Prediction	Learning to predict future feedback as a function of current and recent inputs by temporal difference methods; projection to future states not attempted due to lack of causal models
	Dynamic Decision Making	Using dynamic programming and its variants to learn dynamic strategies from evaluative feedback for Markovian representations
	Nondeterminism	Controlled using stochastic strategies
H	Ill-posed Objective Functions	Off-line learning of interval-evaluated objective functions from absolute or relative evaluation
	Credit Assignment, Storing Past Decisions, Non-Markovian Representations	Off-line learning of causal and temporal models, explicit storage of past decisions (with size limited by temporal scopes), and combination of causal and temporal scopes; valid even for non-Markovian representations
P	Credit Assignment	Maintenance of a population of competing strategies, and use of credit assignment only to alter probability of selection but not necessarily to improve the incumbent strategy; avoidance of structural credit assignment entirely, hence, suitable for decision procedures

K: Knowledge-based (Fig. 6a); R: Reinforcement-learning (Fig. 6b); H: Hybrid point-based (Fig. 6c); P: Population-based (Fig. 6d)

Table 10: Examples of strategy-learning problems and architectures.

Example	Model	Comments
Load Balancing	K	Knowledge-based critic needs a well-defined performance standard, which is not available. This model lacks mechanisms for predicting future values of time-varying status variables because it assumes that nothing changes outside the problem solver's control.
	R	The adaptive critic attempts to compute eligibility on-line, which does not work for non-Markovian representations. The adaptive performance element is efficient for learning stochastic strategies from evaluative feedback.
	H	The model is suitable for non-Markovian states and time-varying status variables. It is difficult to find a persistence model that properly accounts for feedback delays.
	P	The model obviates the credit-assignment problem. Population-based learning allows multi-pronged performance directed search in strategy space. If background knowledge were available, it can be used in intelligent generation of new strategies.
Ship Steering	K	A good match because of well-defined objective function, static strategies, and knowledge of operator semantics.
	R	The model is unsuitable because of non-Markovian representations, complex decision rules, and deterministic strategies.
	H	Unnecessary overhead is incurred due to the additional functionality for handling dynamic strategies and dynamic history management using temporal scopes.
	P	The functionality for simultaneous testing of multiple strategies is not needed for this problem. Sufficient background knowledge is available so that point-based learning using Dietterich and Buchanan's model requires only a few problem-solving episodes.
Pole Balancing	K	Knowledge-based learning is not applicable to the ill-defined objective function and time-varying problem variables.
	R	A perfect match for this problem due to Markovian representation, ill-defined objective function measurable at instants, knowledge-lean environment, and evaluative feedback.
	H	This model works but fails to exploit Markov property for efficient solutions to SCA and TCA.
	P	Point-based learning suffices since Markov property is satisfied and TCA can be solved efficiently.
Towers of Hanoi	K	Perfect match for this model due to its ability to handle multiple, general objectives. Syntactic generalization techniques of this model work well for constrained generalization of observed solutions.
	R	The model lacks mechanisms for handling hard constraints and learning from complete solutions.
	H	Since the Markovian property is satisfied, the extra functionality of this model is not used.
	P	Lengthy test-cases preclude the application of population-based learning.

K: Knowledge-based (Fig. 6a); R: Reinforcement-learning (Fig. 6b); H: Hybrid point-based (Fig. 6c); P: Population-based (Fig. 6d) and knowledge-lean domain.

A clean separation of issues from applications allows us to study strategy-learning techniques independent of the specific learning system in which they are embedded. General-purpose approaches, such as dynamic programming, regression, and time-series analysis, are shown to be useful. Non-Markovian problems, for which no general purpose techniques are known, require new approaches to update dynamically history information and perform rational credit assignment on stored solutions.

The numerous issues and approaches can be abstracted into four general architectures: the knowledge-based learning model, the reinforcement-learning model, the hybrid point-based learning model and the population-based learning model. The first is useful for learning static strategies in knowledge-rich environments; the second, for learning dynamic strategies for Markovian problems in knowledge-lean environments; the third, for learning dynamic strategies for non-Markovian problems; and the fourth, for resource-constrained learning of procedurally encoded strategies using generate-and-test approaches. The first three employ point-based learning, while the fourth one employs population-based learning. The architectures reviewed in Section 4 are by no means exhaustive; for example, none of the architectures work too well in real-time systems where both resource-constrained and on-line learning are desired.

Certain issues in strategy learning are relevant only to specific techniques, such as the slowness of reinforcement learning, the difficulty in generalizing the structure of strategies in explanation-based learning, and the difficulty of methods for non-linear prediction and regression. We list below a few important general issues that need further research.

a) *On-line versus Off-line Learning.* This distinction has been ignored in our classification primarily because several learning techniques have both on-line and off-line versions. On-line learning algorithms can adapt under non-stationary or time-varying environments, and have lower memory requirements. However, they are slower and are harder to design because of the greater efficiency desired of them. Even though numerous on-line learning procedures are known, their general principles are not well understood.

b) *Scaling and Generalization across Tasks.* Scaling requires that the strategies acquired should generalize to problem sizes different from those used in training. Generalization across tasks requires that the strategies acquired should generalize to tasks other than those used for training. This direction of research is well represented by the work of Singh [142] and Tesauro [30].

c) *Exploration-Convergence Dilemma.* At what rate should the randomness of stochastic strategies be reduced so that the performance trace is considered sufficient for convergence to the best strategy so far? This issue reappears as the trade-off between generation (sampling) and testing (confidence-building) in population-based architectures. Add in deadlines and the general problem is beyond the scope of existing theory. The challenge is to get a rational solution to the trade-offs in question without making unreasonable restrictive assumptions. Thrun [143] examines several techniques for addressing this issue.

d) *Quality-Efficiency Trade-off.* We have ignored the cost of applying a strategy in our discussion. However, we have considered both the time and resources needed for learning as well as the cost of the solutions found by the strategy. How should one choose between a strategy that finds a poor solution quickly versus another that takes a long time to find a good solution? Credit-assignment procedures may suggest different changes for improving quality than for improving efficiency. Wefald and Russell [144] suggest a decision-theoretic approach to rational resolution of this trade-off.

Acknowledgments

Research was supported partially by National Science Foundation Grants MIP 92-18715 and MIP 96-32316, and by National Aeronautics and Space Administration Contract NAG 1-613.

The second author would like to thank Prof. H. Tanaka and Fujitsu Limited, who provided support when the author was visiting the University of Tokyo, and the Chinese University of Hong Kong during which part of this paper was written.

References

[1] A. Barr and E. A. Feigenbaum, *The Handbook of Artificial Intelligence*, vol. 1, 2, and 3. Los Altos, CA: William Kaufmann, (1981).

[2] J. Pearl, *Heuristics–Intelligent Search Strategies for Computer Problem Solving.* Reading, MA: Addison-Wesley, (1984).

[3] M. H. Romanycia and F. Pelletier, *What is a heuristic?*, Computational Intelligence **1** National Research Council Canada (1985) 47–58.

[4] M. P. Georgeff, *Strategies in heuristic search*, Artificial Intelligence **20** North Holland (1983) 393–425.

[5] R. M. Keller, *A survey of research in strategy acquisition*, Tech. Rep. DCS-TR-115, Dept. of Computer Science, Rutgers Univ., New Brunswick, NJ, May 1982.

[6] T. M. Mitchell, *Learning and problem solving*, Proc. 8th Int'l Joint Conf. on Artificial Intelligence, (Los Altos, CA), William Kaufman Aug. 1983 1139–1151.

[7] N. S. Sridharan and J. L. Bresina, *Exploration of problem reformulation and strategy acquisition: A proposal*, Tech. Rep. LCSR-TR-53, Lab. Computer Science Research, Rutgers Univ., New Brunswick, NJ, Mar. 1984.

[8] P. Langley, *Learning to search: From weak methods to domain-specific heuristics*, Cognitive Science **9** Ablex Pub. Co. (1985) 217–260.

[9] M. Minsky, *Steps toward artificial intelligence*, Computers and Thought, E. A. Feigenbaum and J. Feldman, eds., New York: McGraw-Hill (1963) 406–450.

[10] R. S. Sutton, *Temporal credit assignment in reinforcement learning*, Ph.D. thesis, Univ. of Massachusetts, Amherst, MA, Feb. 1984.

[11] T. M. Mitchell, *Toward combining empirical and analytical methods for inferring*, Artificial and Human Intelligence, Banerji and Elithorn, eds., New York: Elsevier (1984) 81–103.

[12] S. W. Wilson, *Hierarchical credit allocation in classifier systems*, Genetic Algorithms and Simulated Annealing, L. Davis, ed., Research Notes in Artificial Intelligence, London: Pitman (1987)

[13] G. F. DeJong and R. J. Mooney, *Explanation-based learning: An alternative view*, Machine Learning **1**, no. 2 Kluwer Academic Pub. (1986) 145–176.

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

[15] K. F. Lee and S. Mahajan, *A pattern classification approach to evaluation function learning*, Artificial Intelligence **36** North-Holland (1988) 1–25.

[16] P. Schooley, *Learning state evaluation functions*, Machine Learning Kluwer Academic Pub. (1985) 177–179.

[17] J. Christensen and R. E. Korf, *A unified theory of heuristic evaluation functions and its application to learning*, Proc. National Conf. on Artificial Intelligence, AAAI, Inc. (1986) 148–152.

[18] G. Tesauro, *Connectionist learning of expert backgammon evaluations*, Machine Learning Kluwer Academic Pub. (1988) 200–206.

[19] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., *Machine Learning: An Artificial Intelligence Approach*, vol. 1, 2. Los Altos, CA: William Kaufmann, (1983).

[20] W. L. Brogan, *Modern Control Theory*. Englewood Cliffs, NJ: Prentice-Hall, (1985).

[21] A. G. Barto, S. J. Bradtke, and S. P. Singh, *Real-time learning and control using asynchronous dynamic programming*, Tech. report 91-57, Dept. of Computer Sc., Univ. of Massachusetts, Amherst, MA, (1991).

[22] B. Chandrasekaran, *Towards a taxonomy of problem solving types*, AI magazine Winter/Spring 1983 9–17.

[23] T. G. Dietterich and B. G. Buchanan, *The role of critic in learning systems*, Tech. Rep. STAN-CS-81-891, Stanford Univ., CA, Dec. 1981.

[24] T. Dean and K. Kanazawa, *Probabilistic temporal reasoning*, Proc. National Conf. on Artificial Intelligence AAAI-88 (1988) 524–528.

[25] W. W. S. Wei, *Time Series Analysis: Univariate and Multivariate Methods*. Redwood City, CA: Addison-Wesley, (1990).

[26] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing*. Boston, MA: Kluwer Academic Pub., (1987).

[27] A. G. Barto, R. S. Sutton, and C. W. Anderson, *Neuronlike adaptive elements that can solve difficult learning control problems*, Trans. on Systems, Man and Cybernetics **SMC-13**, no. 5 IEEE (1983) 834–846.

[28] A. L. Samuel, *Some studies in machine learning using the game of checkers*, IBM J. Research and Development **3** IBM (1959) 210–229.

[29] A. L. Samuel, *Some studies in machine learing using the game of checkers II–recent progress*, J. of Research and Development **11**, no. 6 IBM (1967) 601–617.

[30] G. Tesauro, *Practical issues in temporal difference learning*, Machine Learning **8**, no. 3/4 (Special Issue on Reinforcement Learning) Kluwer Academic Publishers (1991) 257–278.

[31] J. E. Laird, P. S. Rosenbloom, and A. Newell, *Soar: An architecture for general intelligence*, Artificial Intelligence **33**, no. 1 Elsevier Science Pub. (1987) 1–64.

[32] J. Laird, P. Rosenbloom, and A. Newell, *Chunking in SOAR: The anatomy of a general learning mechanism*, Machine Learning **1**, no. 1 Kluwer Academic Pub. (1986) 11–46.

[33] R. Barletta and R. Kerber, *Improving explanation-based indexing with empirical learning*, Machine Learning Kluwer Academic Pub. (1989) 84–86.

[34] S. Whitehead and D. Ballard, *A role for anticipation in reactive systems that learn*, Proc. 6th Int'l. Workshop on Machine Learning, (San Mateo, CA), Morgan Kaufmann (1989) 354–357.

[35] R. Mirchandaney and J. A. Stankovic, *Using stochastic learning automata for job scheduling in distributed processing systems*, J. Parallel and Distributed Computing Academic Press (1986) 527–552.

[36] P. Mehra and B. W. Wah, *Learning load-balancing strategies using artificial neural networks*, Intelligent Engineering Systems through Artificial Neural Networks (Proc. Int'l Conf. on Artificial Neural Networks in Engineering) ASME Press (1991) 855–860.

[37] G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control, 2nd ed.* San Francisco: Holden-Day, (1976).

[38] P. Mehra, *Automated Learning of Load Balancing Strategies for a Distributed Computer System*. Urbana, IL: Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Dec. 1992.

[39] D. Ferrari, *A study of load indices for load balancing schemes*, Workload Characterization of Computer Systems and Computer Networks, G. Serazzi, ed., Amsterdam, Netherlands: Elsevier Science (1986) 91–99.

[40] S. Zhou, *Performance studies of dynamic load balancing in distributed systems*, Tech. Rep. UCB/CSD 87/376 (Ph.D. Dissertation), Computer Science Division, Univ. of California, Berkeley, CA, (1987).

[41] P. G. Hoel, S. Port, and C. J. Stone, *Introduction to Stochastic Processes.* Atlanta, GA: Houghton Mifflin Co., (1972).

[42] R. Bellman and S. Dreyfus, *Applied Dynamic Programming.* Princeton, NJ: Princeton Univ. Press, (1962).

[43] D. Ferrari and S. Zhou, *A load index for dynamic load balancing*, Proc. Fall Joint Computer Conf., ACM/IEEE Nov. 1986 684–690.

[44] P. Mehra and B. W. Wah, *Adaptive load-balancing strategies for distributed systems*, Proc. 2nd Int'l Conf. on Systems Integration, (Morristown, NJ), IEEE Computer Society June 1992 666–675.

[45] E. Rich and K. Knight, *Artificial Intelligence.* New York: McGraw Hill, (1991).

[46] C.-T. Chen, *Linear System Theory and Design.* New York: Holt, Rinehart and Winston, Inc., (1970).

[47] N. J. Nilsson, *Principles of Artificial Intelligence.* Tioga, (1980).

[48] E. Sacerdoti, *The nonlinear nature of plans*, Reasoning about Actions and Plans, M. Georgeff and A. Lansky, eds., Morgan Kaufmann (1987) 206–214.

[49] C. W. Anderson, *Strategy learning with multilayer connectionist representations*, Proc. Fourth Int'l. Workshop on Machine Learning, Morgan Kaufmann June 1987 103–114.

[50] K. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction.* Englewood Cliffs, NJ: Prentice Hall, (1989).

[51] M. P. Georgeff, *Planning*, Annual Review of Computer Science, vol. 2, Palo Alto, CA: Annual Reviews Inc. (1987) 359–400.

[52] L. Kaebling, *An architecture for intelligent reactive systems*, Reasoning about Actions and Plans, M. Georgeff and A. Lansky, eds., Los Altos, CA: Morgan Kaufmann (1987)

[53] M. J. Schoppers, *Representation and automatic synthesis of reaction plans*, Ph.D. thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL., (1988).

[54] P. S. Ow, S. F. Smith, and A. Thiriez, *Reactive plan revision*, Proc. Tenth National Conf. on Artificial Intelligence AAAI-88, vol. 1, (Saint Paul, MN) (1988) 77–82.

[55] M. P. Georgeff and A. L. Lansky, *Reactive reasoning and planning*, Proc. National Conf. on Artificial Intelligence, (Seattle, Washington), AAAI, Inc. June 1987 677–82.

[56] P. E. Agre and D. Chapman, *PENGI: An implementation of a theory of activity*, Proc. National Conf. Artificial Intelligence, (Palo Alto, CA), Morgan Kaufman June 1987 268–272.

[57] P. Maes, *How to do the right thing*, Connection Science **1**, no. 3 (1989) 291–323.

[58] C. Gao, J. W. S. Liu, and M. Railey, *Load balancing algorithms in homogeneous distributed systems*, Proc. Int'l Conf. Parallel Processing, IEEE Aug. 1984 302–306.

[59] R. M. Bryant and R. A. Finkel, *A stable distributed scheduling algorithm*, Proc. 1st Int'l Conf. on Distributed Computing Systems, IEEE (1981) 314–323.

[60] K. Baumgartner and B. W. Wah, *GAMMON: A load balancing strategy for a local computer system with a multiaccess network*, IEEE Trans. on Computers **38** IEEE Aug. 1989 1098–1109.

[61] M. L. Litzkow, M. Livny, and M. W. Mutka, *Condor - A hunter of idle workstations*, Proc. 8th Int'l. Conf. Distributed Computer Systems, IEEE (1988) 104–111.

[62] G. E. Hinton, *Connectionist learning procedures*, Artificial Intelligence **40** Elsevier Science Pub. (1989) 185–234.

[63] B. Widrow, N. K. Gupta, and S. Maitra, *Punish/reward: Learning with a critic in adaptive threshold systems*, Trans. Systems, Man, and Cybernetics **SMC-3**, no. 5 IEEE (1973) 455–465.

[64] A. Ieumwananonthachai, A. N. Aizawa, S. R. Schwartz, B. W. Wah, and J. C. Yan, *Intelligent mapping of communicating processes in distributed computing systems*, Proc. Supercomputing 91, (Albuquerque, NM), ACM/IEEE Nov. 1991 512–521.

[65] D. Nguyen and B. Widrow, *The truck backer-upper: An example of self-learning in neural networks*, Proc. Int'l Joint Conf. on Neural Networks, vol. II, IEEE (1989) 357–363.

[66] K. K. Goswami and R. K. Iyer, *Dynamic load-sharing using predicted process resource requirements*, Tech. Rep. UILU-ENG-90-2224, Coordinated Sci. Lab., Univ. of Illinois, Urbana, (1990).

[67] B. Porter and D. Kibler, *Experimental goal regression: A method for learning problem-solving heuristics*, Machine Learning **1**, no. 3 Kluwer Academic Pub. (1986) 249–286.

[68] R. A. Howard, *Dynamic Programming and Markov Processes.* London: Jon Wiley, (1960).

[69] R. S. Sutton, *Convergence theory for a new kind of prediction learning,* Proc. 1988 Workshop on Computational Learning Theory (D. Haussler and L. Pitt, eds.), (Palo Alto, CA), Morgan Kaufmann (1988) 421–422.

[70] R. S. Sutton, *Learning to predict by the methods of temporal differences,* Machine Learning **3** Kluwer Academic Pub. Aug. 1988 9–44.

[71] 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 **15** IEEE Apr. 1982 55–66. Also in *Tutorial: Computer Architecture,* ed. D. D. Gajski, V. M. Milutinovic, H. J. Siegel and B. P. Furht, IEEE Computer Society, 1987, pp. 541-552.

[72] Y. Anzai, *Doing, understanding, and learning in problem solving,* Production System Models of Learning and Development, *et al.* Klahr, ed., Cambridge, MA: MIT Press (1987)

[73] R. E. Korf, *Macro-operators: A weak method for learning,* Artificial Intelligence **26** North-Holland (1985) 35–77.

[74] P. M. Andreae, *Constraint limited generalization: Acquiring procedures from examples,* Proc. National Conf. Artificial Intelligence, (Austin, TX), AAAI, Inc. (1984) 6–10.

[75] P. Mehra and B. W. Wah, *Architectures for strategy learning,* Computer Architectures for Artificial Intelligence Applications, B. Wah and C. Ramamoorthy, eds., New York, NY: Wiley (1990) 395–468.

[76] D. E. Goldberg, *Probability matching, the magnitude of reinforcement, and classifier system bidding,* Machine Learning **5** Kluwer Academic Pub. (1990) 407–425.

[77] L. A. Rendell, *An adaptive plan for state-space problems,* Tech. Rep. CS-81-13, Univ. of Waterloo, Ontario, Canada, Mar. 1981.

[78] SAS Institute, Inc., *SAS/ETS user's guide, version 5 edition,* (1984).

[79] T. W. Mirer, *Economic Statistics and Econometrics.* New York: Macmillan, (1983).

[80] L.-J. Lin and T. M. Mitchell, *Memory approaches to reinforcement learning in non-markovian domains (tech. report no.,* Technical Report CMU-CS-92-138), Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, May 1992.

[81] P. Munro, *A dual back-propagation scheme for scalar reward learning,* Proc. Ninth Annual Conf. of the Cognitive Science Society, (Hillsdale, NJ), Lawrence Erlbaum Associates (1987) 165–176.

[82] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, eds., vol. 1, Cambridge, MA: MIT Press (1986) 318–362.

[83] C. F. Yu, *Efficient Combinatorial Search Algorithms.* West Lafayette, IN: Ph.D. Thesis, School of Elect. Engr., Purdue Univ., Dec. 1986.

[84] M. Gluck, D. B. Parker, and E. S. Reifsnider, *Learnin[illegible] temporal deivatives in pulse-coded neuronal systems*, Proc. Neural Information Processing Systems (D. Z. Anderson, ed.), (New York), American Inst. of Physics (1988)

[85] J. S. Morgan, E. C. Patterson, and A. H. Klopf, *Drive-reinforcement learning: A self-supervised model for adaptive control*, Network: Computation in Neural Systems **1** IOP Pub. Ltd. (1990) 439–448.

[86] G. Tesauro and T. J. Sejnowski, *A parallel network that learns to play backgammon*, Artificial Intelligence **39** Elsevier Science Pub. (1989) 357–390.

[87] P. Mehra and B. W. Wah, *Automated learning of workload measures for load balancing on a distributed system*, Proc. Int'l Conference on Parallel Processing, CRC Press Aug. 1993 III–263–III–270.

[88] M. R. Genesereth and N. J. Nilsson, *Logical Foundations of Artificial Intelligence.* Los Altos, CA: Morgan Kaufmann, (1987).

[89] T. Mitchell, R. Keller, and S. Kedar-Cabelli, *Explanation-based generalization: A unifying view*, Machine Learning **1**, no. 1 Kluwer Academic Pub. (1986) 47–80.

[90] 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 (T. G. Dietterich, ed.), Computer Science Dept., Oregon State Univ. (1986) 77–87.

[91] B. W. Wah, *Population-based learning: A new method for learning from examples under resource constraints*, IEEE Trans. on Knowledge and Data Engineering **4** IEEE Oct. 1992 454–474.

[92] A. Ieumwananonthachai, A. Aizawa, S. R. Schwartz, B. W. Wah, and J. C. Yan, *Intelligent process mapping through systematic improvement of heuristics*, J. of Parallel and Distributed Computing **15** Academic Press June 1992 118–142.

[93] P. Mehra and B. W. Wah, *Population-based learning of load balancing policies for a distributed computer system*, Proc. Computing in Aerospace 9 Conf., (San Diego, CA), American Institute of Aeronautics and Astronautics Oct. 1993 1120–1130.

[94] B. W. Wah and H. Kriplani, *Resource constrained design of artificial neural networks*, Proc. Int'l Joint Conf. on Neural Networks, vol. III, IEEE June 1990 269–279.

[95] C.-C. Teng and B. W. Wah, *Mixed-mode learning: A method for reducing the number of hidden units in cascade correlation*, Proc. Int'l Symposium on Artificial Neural Networks, (Hsinchu, Taiwan), National Chiao Tung University Dec. 1993 I–01–I–07.

[96] M. Lebowitz, *Integrated learning: Controlling explanation*, Cognitive Science Ablex Pub. Co. (1986) 219–240.

[97] R. S. Sutton and B. Pinette, *The learning of world models by connectionist networks*, Proc. Seventh Ann. Conf. Cognitive Science Soc., (Hillsdale, NJ), Lawrence Erlbaum Associates (1985) 54–64.

[98] H. A. Kautz, *The logic of persistence*, Proc. National Conf. on Artificial Intelligence, Morgan Kaufman (1986) 401.

[99] L. Morgenstern and L. A. Stein, *Why things go wrong: A formal theory of causal reasoning*, Proc. National Conf. on Artificial Intelligence AAAI-88 (1988) 518–523.

[100] T. Dean and K. Kanazawa, *A model for reasoning about persistence and causation*, Compuational Intelligence **5**, no. 3 National Research Council Canada (1989) 142–150.

[101] C. W. Anderson, *Learning and problem solving with multilayer connectionist systems*, Ph.D. thesis, Univ. of Massachusetts, Amherst, MA, (1986).

[102] J. H. Schmidhuber, *Making the world differentiable,*, Tech. Rep. FKI-126-90, Technical Univ. of Munich, Munich, Germany, (1990).

[103] M. I. Jordan, *Supervised learning and systems with excess degrees of freedom*, Proc. Connectionist Models Summer School (D. Touretzky, G. E. Hinton, and T. J. Sejnowski, eds.), (Palo Alto, CA), Morgan Kaufmann (1988) 62–75.

[104] T. K. Miller, III, R. S. Sutton, and P. J. Werbos, eds., *Neural Networks for Control.* Cambridge, MA: MIT Press, (1990).

[105] M. Kendall and J. K. Ord, *Time Series.* London: 3rd ed., I Edward Arnold, (1990).

[106] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart, *Predicting the future: A connectionist approach*, Int'l. J. of Neural Systems **1**, no. 3 World Scientific Pub. (1990) 209.

[107] K. J. Lang and G. E. Hinton, *A Time-Delay Neural Network Architecture for Speech Recognition.* Pittsburgh, PA: CMU-CS-88-152, Dept. of Computer Science, Carnegie Mellon Univ., (1988).

[108] R. K. Mehra, *Kalman filters and their applications to forecasting*, TIMS Studies in Management Sciences **12** North-Holland (1979) 75–94.

[109] G. C. Goodwin and K. S. Sin, *Adaptive Filtering Prediction and Control.* Englewood Cliffs, NJ: Prentice-Hall, (1984).

[110] S. Hanks, *Practical temporal projection*, Proc. 8th Natl. Conf. Artificial Intelligence, (Seattle, Washington), AAAI, Inc. (1990) 158–163.

[111] T. Dean and G. Siegle, *An approach to reasoning about continuous change for applications in planning*, Proc. 8th Natl. Conf. Artificial Intelligence, (Seattle, Washington), AAAI, Inc. (1990) 132–137.

[112] S. A. Chien, *Learning by analyzing fortuitous occurances*, Machine Learning Kluwer Academic Pub. (1989) 249–251.

[113] S. A. Chien, *An explanation-based learning approach to incremental planning*, Ph.D. thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, (1991).

[114] D. White, *Dynamic Programming.* Edinburgh, UK: Oliver and Boyd, (1969).

[115] P. J. Werbos, *Consistency of HDP applied to a simple reinforcement learning problem*, Neural Networks **3** Pergamon Press (1990) 179–189.

[116] C. F. Yu and B. W. Wah, *Learning dominance relations in combinatorial search problems*, IEEE Trans. on Software Engineering **SE-14** IEEE Aug. 1988 1155–1175.

[117] T. Ibaraki, *Enumerative approach to combinatorial optimization*, Annals of Operations Research Scientific Pub. Co. (1988)

[118] S. Ross, *Introduction to Stochastic Dynamic Programming.* New York: Academic Press, (1983).

[119] P. J. Werbos, *A menu of designs for reinforcement learning over time*, Neural Networks for Control, Miller et al., ed., Cambridge, MA: MIT Press (1990) 67–96.

[120] G. R. Walsh, *Methods of Optimization.* London, England: Wiley, (1977).

[121] B. A. Nadel, *Constraint saisfaction algorithms*, Computational Intelligence **5** National Research Council Canada (1989) 188–224.

[122] D. McAllester, *Truth maintenance*, Proc. 8th Natl. Conf. Artificial Intelligence, (Seattle, Washington), AAAI, Inc. (1990) 1109–16.

[123] B. W. Mel, *MURPHY: A robot that learns by doing*, Neural Information Processing Systems, D. Z. Anderson, ed., New York, NY: American Institute of Physics (1988) 544–553.

[124] T. G. Dietterich and J. S. Bennett, *The test incorporation theory of problem solving*, Proc. Workshop on Knowledge Compilation, Dept. of Computer Science, Oregon State Univ., Sept. 1986 145–159.

[125] R. S. Sutton, *Integrated architectures for learning, planning, and reacting based on approximating dynamic programming*, Proc. 7th Int'l. Conf. Machine Learning, (Palo Alto, CA), Morgan Kaufmann (1990) 216–224.

[126] J. G. Carbonell, *Learning by analogy: Formulating and generalizing plans from past experiences*, Machine Learning, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Tioga (1983)

[127] R. Greiner, *Learning by understanding analogies*, Machine Learning Kluwer Academic Pub. (1985) 50–52.

[128] J. H. Holland, *Properties of the bucket brigade algorithm*, Proc. Int'l. Conf. Genetic Algorithms and Their Applications (J. J. Grefenstette, ed.), (Pittsburgh, PA), The Robotics Inst. of Carnegie-Mellon Univ. (1985) 1–7.

[129] C. Lewis, *Composition of productions*, Production System Models of Learning and Development, Klahr and *et al.*, eds., Cambridge, MA: MIT Press (1987)

[130] P. Rosenbloom and A. Newell, *Learning by chunking: A production system model of practice*, Production System Models of Learning and Development, Klahr and *et al.*, eds., Cambridge, MA: MIT Press (1987)

[131] R. Neches, *Learning through incremental refinement of procedures*, Production System Models of Learning and Development, Klahr and *et al.*, eds., Cambridge, MA: MIT Press (1987)

[132] D. Steier, J. Laird, A. Newell, and P. Rosenbloom, *Varieties of learning in SOAR: 1987*, Machine Learning Kluwer Academic Pub. (1987) 300–311.

[133] R. Michalski, I. Mozetic, J. Hong, and N. Lavrac, *The multi-purpose incremental learning system AQ15 and its testing applications to three medical domains.*, Proc. of the Fifth National Conference on Artificial Intelligence, (Philadelphia, PA), Morgan Kaufmann (1986) 1041–1045.

[134] T. M. Mitchell, *Generalization as search*, Artificial Intelligence **18** North Holland (1982) 203–226.

[135] 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, (Los Altos, CA), William Kaufman Aug. 1977 338–343.

[136] L.-J. Lin, *Self-improving reactive agents based on reinforcement learning, planning, and teaching*, Machine Learning **8**, no. 3/4 (Special Issue on Reinforcement Learning) Kluwer Academic Publishers (1992) 293–322.

[137] S. F. Smith, *Flexible learning of problem solving heuristics through adaptive search*, Proc. Int'l Joint Conf. on Artificial Intelligence, Morgan Kaufman (1983) 422–5.

[138] L. B. Booker, D. E. Goldberg, and J. H. Holland, *Classifier systems and genetic algorithms*, Machine Learning: Paradigm and Methods, J. Carbonell, ed., MIT press (1990)

[139] B. K. Ghosh and P. K. Sen, eds., *Handbook of Sequential Analysis.* New York, NY: Marcel Dekker, Inc., (1991).

[140] L.-C. Chu, *Algorithms for Combinatorial Optimization in Real Time and their Automated Refinement by Genetic Programming.* Urbana, IL: Ph.D. Thesis, Dept. of Electrical and Computer Engineering, Univ. of Illinois, May 1994.

[141] S. R. Schwartz, *Resource Constrained Parameter Tuning Applied to Stereo Vision.* Urbana, IL: M.Sc. Thesis, Dept. of Electrical and Computer Engineering, Univ. of Illinois, Aug. 1991.

[142] S. P. Singh, *Transfer of learning by composing solutions of elemental tasks,* Machine Learning 8, no. 3/4 (Special Issue on Reinforcement Learning) Kluwer Academic Publishers (1992) 323–340.

[143] S. B. Thrun, *Efficient exploration in reinforcement learning,* Tech. Rep. CMU-CS-92-102, School of Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA, Jan. 1992.

[144] S. Russell and E. Wefald, *Principles of metareasoning,* Artificial Intelligence **49** Elsevier (1991) 361–395.