Issues in Strategy Learning

Benjamin W. Wah
Center for Reliable and High Performance Computing
Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
1308 West Main Street
Urbana, IL 61801, USA
wah@manip.crhc.uiuc.edu

Abstract - In this paper, we discuss twelve application-specific issues in strategy learning. These issues are important when designing strategy learning systems as well as in learning new application-specific strategies. In each issue we discuss the problems involved and identified some effective solutions.

I. INTRODUCTION

In the last five years, we have witnessed a significant improvement in computational power with the introduction of massively parallel processing systems. An example of such an effort is the Federal High Performance Computing and Communications Initiative (HPCCI) [14] that aims at designing computers with tera-flops substained performance by the year 2000. These computers are targeted to solve computationally-intensive problems in areas such as vision and cognition, superconductor modeling, speech and natural language processing, and human genome. The nature of these applications is that they are not bounded in their size; additional computing power can always be used to solve larger problems. Consequently, it is essential to develop better methods to solve these application problems, as well as to find efficient ways of mapping these applications on new massively parallel processing systems.

Machine learning is one of the effective methods to automatically acquire heuristics for problem solving as well as for mapping computations on high-performance computing systems. *Heuristics*, in this context, are modular pieces of readily applicable knowledge — "rules of thumb" [12] — 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 [6]. 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 [12].

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.

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 the class of problems whose instances are to be solved. Ill-posed learning problems have performance tasks with *ill-defined objective functions*; that is, the objective 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).

Research supported by National Science Foundation Grant MIP 92-18715 and National Aeronautics and Space Administration Contract NAG 1-613.

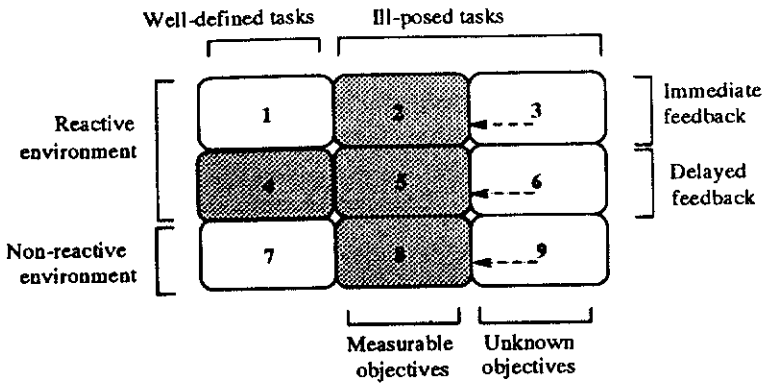

Figure 1. Classification of strategy-learning problems (Shaded areas represent problems of interest.)

The class of problems of interest are 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.

II. Twelve Problem-Specific Issues in Strategy Learning

In this section we present concisely twelve problem-specific issues for strategy learning. In each issue, we first define the problem and discuss a few solution methods. A more detailed survey can be found in the reference [10].

1) *Ill-posedness of 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.

For application problems 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 9 of Figure 1), feedback must be generated internally by the learning system.

The following issues arise in learning strategies when objective functions are ill-defined.

a) *Standard-of-comparison problem.* This problem, first recognized by Ackley [1], concerns the method of assessing feedback. If the objective function of an application problem is ill-defined, then it is difficult to assess solution quality in absolute terms, and alternative operators can only be evaluated relative to each other. In this case, learning can be used to improve an existing strategy, and the notion of optimal strategy is undefined.

b) *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.

c) *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 [3]. Learning while searching is also useful for problem solvers that employ deterministic strategies to expand large search spaces [13]. 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.

2) *Inconsistencies in performance evaluation.* Inconsistencies often happen when objective functions are learned. Consider an application with two strategies S_1 and S_2 and two sets of test cases T_1 and T_2. For two objective functions $\mathbf{O_1}$ and $\mathbf{O_2}$, four combinations of objectives and test cases are possible: i) $\mathbf{O_1}$ and T_1, ii) $\mathbf{O_1}$ and T_2, iii) $\mathbf{O_2}$ and T_1, and iv) $\mathbf{O_2}$ and T_2. In each of these combinations, S_1 can be better than or worse than S_2. Hence, inconsistent conclusions may be drawn depending on the order the test cases are experimented and the objective functions are formulated. In general, there can be infinitely many possible objective functions and test cases of different behavior for a given application. Further, inconsistencies are possible depending on the particular strategy chosen as the baseline strategy for improvement, and the range of performance values obtained for different test cases.

To address the above inconsistencies, we have studied two methods. i) For application problems whose test cases have similar statistical behavior as far as performance is concerned [15], we need to first identify a reference or baseline strategy and normalize each performance attribute of a new strategy consistently with respect to the same attribute of the reference strategy. This avoids the problem in which one strategy is better than another under one baseline strategy, but worse under another baseline strategy. Second, two strategies should be compared using individual normalized performance attributes rather than using a single objective function. A possible way when the number of attributes is small is to represent the performance values using a multi-dimensional scatter plot. In this case, the performance of a strategy on a test case is represented as a point in a multi-dimensional plot whose axes are the performance attributes of the application. This avoids inconsistencies when objectives are formed arbitrarily as functions of performance attributes. ii) For application problems whose test cases have different statistical behavior, we need to first partition the space of test cases into subdomains [16]. This step may have to be done heuristically based on available domain knowledge on the application. Strategies are then learned for a subset of the subdomains in the way described in (i) above. After learning is completed, strategies learned for each subdomain are evaluated on all remaining subdomains to determine the largest cluster of subdomains that have similar performance and the best strategy to be used for each cluster. In evaluating performance of strategies across different subdomains, a measure independent of the range of performance values may need to be used. An example of such a measure is the probability of win which identifies a strategy that wins over other strategies in multiple subdomains.

3) *Credit assignment.* This deals with methods for using feedback to correct errors in the decision process. Feedback can be either a corrective error signal or a scalar evaluation signal, covering

either one or more decision points, and generated either periodically or intermittently. Different types of feedback require different schemes for credit assignment, and consequently, different kinds of learning algorithms.

Feedback can be differentiated as prescriptive or evaluative. 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* [8]. On the other hand, evaluative feedback carries only implicit information about the desired behavior but explicit evaluation of the observed behavior. Generating such feedback requires only a *critic* [20] 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* [11] and learning from such signals, *reinforcement learning*.

There are two types of credit assignment: structural and temporal. Structural credit assignment (SCA) deals with methods for apportioning feedback to elements of the decision process. These methods depend on the 'structure' of the process, and whether the feedback is prescriptive or evaluative. They are more difficult to design when the process is procedural and the feedback, evaluative.

Temporal credit assignment (TCA), in contrast, deals with methods of handling delayed feedback which explicitly evaluates the current state and implicitly evaluates past states and decisions. Based on the temporal properties of feedback, one can distinguish between temporally local and temporally global feedback. The former applies to decisions individually, whereas the latter applies to decisions collectively. Translation of temporally global feedback into temporally local feedback is called temporal credit assignment.

Methods for TCA depends on whether the state space is Markovian. Non-Markovian representations generally require more complex TCA procedures. In this case, past decisions and/or states may need to be retained because they influence the feedback 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.

4) *Prediction*. In contrast to credit assignment, prediction deals with the prediction of future states and future feedback based on time-varying problem variables in order to minimize errors in the strategies learned. 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 [4]. 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 [17].

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. Methods to deal with such cases include approximate models of the feedback-generation mechanism, and projection and time-series analysis.

5) *Violation of the 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 temporal credit assignment. Corollary issues, such as how to 'forget' old or unimportant decisions, must also be addressed. 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. This issues leads to the following issue.

6) *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.

7) *Constraint handling.* Constraints can be classified as hard or soft, and well-defined or ill-defined. Hard constraints impose sharp boundaries on the state space, demarcating feasible solutions from infeasible ones. 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 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.

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) *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 techniques for achieving general goals. The learning system may need to use general-purpose reasoning techniques [9]. In this case, the procedural component of the knowledge stored should be reduced to a minimum; such reduction 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 [2].

In general, it is impossible to expose a strategy-learning system to all possible instances of a performance task, except for the most trivial problems. 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.

9) *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 a set of rules for determining the new

state given an old state and a decision. Note 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).

In these problems, Markovian representations often accompany objective functions that are either well-defined or ill-defined and measured over instants. 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, the optimal solution can be computed using dynamic programming. Similarly, for ill-defined functions, the objective of maximizing the (possibly discounted) sum of future evaluations is also amenable to dynamic programming. Variants of this procedure exist for knowledge-lean environments as well as for problem solvers with stochastic strategies [18]. In the general case when the representation is non-Markovian, no general solution is known.

10) *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 uses explicit representation of structured solutions during credit assignment, and exploits the solution structure 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.

11) *Controlling non-determinism.* The learning task is characterized by a small number of operators in knowledge-lean environments and the generality of operator preconditions in knowledge-intensive environments. A learning system implementing the learning task must be able to 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 using learning-while-searching. It introduces a controlled element of randomness in decision making by only exploring one randomly selected alternative on each visit to a state. If it visits a state several times, it may explore several different alternatives. This is achieved using stochastic strategies, which are popular in strategy-learning systems for knowledge-lean environments [19].

12) *Resource constraints in learning.* Resource scheduling entails the allocation of a limited amount of computational resources for generating and for testing strategies. Resource scheduling algorithms can be static or dynamic. A static scheduling algorithm determines, before learning begins, the number of times the strategies will be modified in the learning process and the number of tests performed on each strategy. Examples of static scheduling algorithms have been studied in genetics-based classifier systems, in which tradeoffs have been evaluated with respect to different number of generate-and-test phases in a fixed amount of time [7] and different overheads in generation [5]. On the other hand, a dynamic resource scheduling algorithm determines dynamically, based on performance statistics obtained during learning, whether a new strategy is to be generated or an existing strategy is to be tested. This decision may depend on: i) whether existing strategies that perform well have been tested to an adequate degree of confidence, ii) whether the performance of all existing strategies falls below a required minimum, iii) whether there are insufficient number of strategies in the population (for learning based on a population of strategies), and iv) whether there are plenty of resources remaining. Dynamic scheduling algorithms are potentially better because they avoid testing strategies that have been found to be poor with some initial tests [15, 16].

III. CONCLUSIONS

In this paper, we have identified twelve issues related to strategy learning. These issues indicate that there is not a single method that is superior for every application problem. Good learning methods often depend on a combination of domain knowledge and the ingenuity of the designers. Massively parallel systems may not be the solution to problems in learning, as 'better' domain knowledge is often far better than brute-force parallelism. Analog hardware, a counterpart to digital hardware that is often overlooked in AI, may allow more efficient storage of knowledge and search.

REFERENCES

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

[2] R. Barletta and R. Kerber, "Improving Explanation-Based Indexing with Empirical Learning," *Machine Learning*, pp. 84-86, Kluwer Academic Pub., Boston, MA, 1989.

[3] 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*, vol. SMC-13, no. 5, pp. 834-846, IEEE, 1983.

[4] T. Dean and K. Kanazawa, "Probabilistic Temporal Reasoning," *Proc. National Conf. on Artificial Intelligence AAAI-88*, pp. 524-528, 1988.

[5] J. M. Fitzpatrick and J. J. Grefenstette, "Genetic Algorithms in Noisy Environments," *Machine Learning*, vol. 3, no. 2/3, pp. 101-120, Kluwer Academic Pub., Boston, MA, Oct. 1988.

[6] M. P. Georgeff, "Strategies in Heuristic Search," *Artificial Intelligence*, vol. 20, pp. 393-425, North Holland, 1983.

[7] J. J. Grefenstette, C. L. Ramsey, and A. C. Schultz, "Learning Sequential Decision Rules using Simulation Models and Competition," *Machine Learning*, vol. 5, pp. 355-381, Kluwer Academic Pub., Boston, MA, 1990.

[8] G. E. Hinton, "Connectionist Learning Procedures," *Artificial Intelligence*, vol. 40, pp. 185-234, Elsevier Science Pub., New York, 1989.

[9] J. E. Laird, P. S. Rosenbloom, and A. Newell, "Soar: An Architecture for General Intelligence," *Artificial Intelligence*, vol. 33, no. 1, pp. 1-64, Elsevier Science Pub., New York, 1987.

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

[11] M. Minsky, "Steps Toward Artificial Intelligence," in *Computers and Thought*, ed. E. A. Feigenbaum and J. Feldman, pp. 406-450, McGraw-Hill, New York, 1963.

[12] J. Pearl, *Heuristics--Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, MA, 1984.

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

[14] Committee on Physical, Mathematical, and Engineering Sciences, Grand Challenges: High Performance Computing and Communications: The FY 1992 U.S. Research and Development Program, Federal Coordinating Council for Science, Engineering, and Technology, Office of Science and Technology Policy, Washington, DC, 1991.

[15] B. W. Wah, "Population-Based Learning: A New Method for Learning from Examples under Resource Constraints," *Trans. on Knowledge and Data Engineering*, vol. 4, no. 5, pp. 454-474, IEEE, Oct. 1992.

[16] B. W. Wah, A. Aizawa, and A. Ieumwananonthachai, "Real-Time Learning of Heuristics," *Trans. on Knowledge and Data Engineering*, IEEE, (under revision) 1993.

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

[18] P. J. Werbos, "Consistency of HDP Applied to a Simple Reinforcement Learning Problem," *Neural Networks*, vol. 3, pp. 179-189, Pergamon Press, Elmsford, NY, 1990.

[19] S.D. Whitehead and D.H. Ballard, "A Role for Anticipation in Reactive Systems that Learn," *Proc. 6th Int'l. Workshop on Machine Learning*, pp. 354-357, Morgan Kaufmann, San Mateo, CA, 1989.

[20] B. Widrow, N. K. Gupta, and S. Maitra, "Punish/Reward: Learning with a Critic in Adaptive Threshold Systems," *Trans. Systems, Man, and Cybernetics*, vol. SMC-3, no. 5, pp. 455-465, IEEE, 1973.