A STUDY OF EMPIRICAL LEARNING FOR AN INVOLVED PROBLEM

Larry Rendell
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Avenue, Urbana, Illinois 61801

Abstract

In real-world domains a concept to be learned may be unwieldy and the environment may be less than ideal. One combination of difficulties occurs if the concept is probabilistic and the learning situation is dynamic. In this case, the data may be noisy and *biased.* These difficulties arise when learning evaluation functions, which can be considered as concepts. A representative problem, the fifteen puzzle, is used to test six different learning systems: some that fit, count, or partition data in *instance,* space; some that optimize measures derived from data in *hypothesis* space; and some that perform *combinations* of such procedures. These six systems are described, tested, and analyzed. From quantitative differences in several experiments, we extract specific properties. By combining two or three kinds of techniques, we gauge the extent to which they complement each other. Combinations of strengths can overcome difficulties in domains that are simultaneously probabilistic, dynamic, noisy, and biased.

1. Introduction

Although concepts and evaluation functions can be learned from examples using various methods, existing techniques are often inadequate for harder problems. If the domain is uncertain or the environment is dynamic [Langley, 1987], a simple induction algorithm may have difficulty. In many cases we may be forced to elaborate old methods, combine them, or develop new ones.

To complicate matters, there are several approaches to choose from. To learn uncertain concepts, we might consider a modification of ID3 [Quinlan, 1986] or some statistical technique [Draper & Smith, 1981]. Or, because concept learning involves the parameterization of descriptions, we might base our method on traditional optimization [Gill et al., 1981]; one such approach uses genetic algorithms [Holland, 1975]. Of course, some methods are known to be especially suited for particular situations; e.g., genetic algorithms can be applied to badly behaved problems having strong nonlinearities. However, many problems and methods are largely unexplored.

The situations considered in this paper include probabilistic concepts, dynamic environments, and noisy and biased data. In an attempt to help develop methods for such cases, we explore a problem that taxes current systems. The problem involves a representation of the fifteen puzzle that can be managed by elaborations of several methods.

The next section defines a general problem of which the fifteen puzzle is a special case. The analysis suggests certain methods. Section 3 describes the methods and discusses their strengths before we evaluate and compare them in Section 4. Finally, Section 5 summarizes implications for empirical learning.

2. The Problem

This section analyzes a representation of the fifteen puzzle, whose solution may be viewed as a special case of concept learning. In this and other problems, the data may be biased, which complicates learning.

2.1. Generalised Concepts

Our basic problem is learning a concept or evaluation function from examples. By definition, a concept is a rule that describes a class of instances. If we represent an instance as a tuple of attributes, a concept will have an associated instance space whose dimensionality is the total number of attributes. As shown in Figure 1(a), an "all-or-none" concept is a binary-valued function over instance space. This diagram shows two attributes; in general a concept is a function over whatever attributes are used to express it. If we allow a concept to be probabilistic (Fig. lb), the function becomes graded: it has values between 0 (certain class exclusion) and 1 (certain class membership). Here we interpret and learn the graded values as probabilities. Hence a concept is a binary or probabilistic *function over instance space.*

Viewing a concept as a function helps to draw parallels among learning methods and the representations they use. As one example, consider an evaluation function H for a state-space problem such as the fifteen puzzle. If x is k-tuple of attributes $(x_1\ x_2,\ \ldots\ x_k)$ representing a state, then H(x) could represent the *probability of finding the goal* (quickly, optimally,

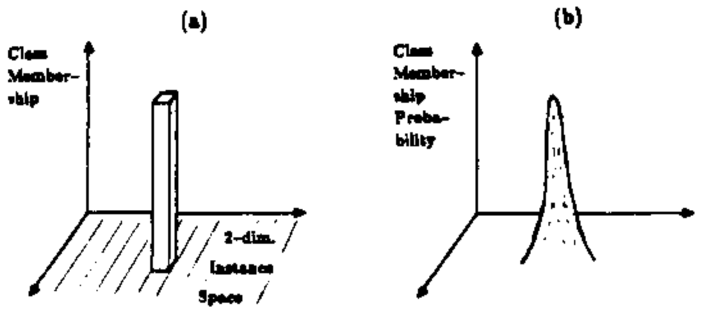

Figure 1. Binary and graded concepts. In (a), the concept is all-or-none; in (b), it is probabilistic. A probabilistic concept is similar to a probabilistic evaluation function.

etc.) *if the solution path goes through* x [Rendell, 1983]. But because H(x) has the form of a probabilistic concept, it could also be interpreted as the concept "good state to develop."

Typically, a concept is expressed as a logic expression or decision tree involving the attributes [Mitchell, 1978; Quinlan, 1986], sometimes with annotated probabilities [Breiman et al., 1984; Rendell, 1983], although other expressions are possible. In contrast, an evaluation function H is often defined as a linear combination of attributes h_1x_1 + b_2x_2 4- .... + b_kx_k = b.x, where the b_i are weights to be learned [Samuel, 1963], However, these preferred forms are not mandatory.

To reiterate: When expressed in terms of attributes, concepts and evaluation functions are both functions over their instance space. Only the details vary. The terms *concept* and *evaluation function* are sometimes synonymous.

2.2. Noisy and Biased Data

The data for empirical learning are often binary, to represent definite knowledge about class membership [Mitchell, 1978]; but data can also be graded, to indicate probable categorization [Draper & Smith, 1981]. For example, we could annotate a value of 0.8 to a patient's state to express a belief that he has some disease. Probabilistic data can also be uncertain (e.g., 0.8 ± 0.1), or even *biased* (e.g., 0.8 is an overestimate). Biased data arise if the sampling is not random, which is undesirable in statistics [Draper & Smith, 1981], but not always in machine learning [Winston, 1984].

In some problems, not all the data are available before some decision must be made that utilizes the results of analyzing the first batch. In such dynamic learning, how might the results of one run affect the *gathering* of future data? A physician might have compiled some data for diagnosis. If these data support disease A, then the physician's subsequent tests may be biased toward observations that will confirm it. The consequences of such a bias may often be good, but sometimes detrimental. If the correct diagnosis is disease B, time may be wasted, or worse, the proper evidence may never be found.

This problem can also arise in domains such as the fifteen puzzle. Because this puzzle has about 10^{13} states, most problems cannot be solved breadth-first (to give unbiased data). In contrast, a good evaluation function solves many problems, but produces biased data: States assessed favorably tend to predominate. Data can become increasingly biased as the evaluation function improves in successive iterations. New data are incommensurable with early data, and if used directly, can give erroneous results [Rendell, 1981].

2.3. Representative Problem Characteristics

The issues explored here involve biased and uncertain data, uncertain concepts, and dynamic learning. All these arise in the fifteen puzzle when an evaluation function is used for best-first search. In experiments, several attributes were defined, all relative to the goal. The most important is the total city-block distance of tiles from their goal positions [Doran & Michie, 1966]. The other attributes are various impediments, such as the tiles in a row being correct, except out of order [Rendell, 1981]. This high-level representation presumes considerable knowledge, and compresses the 10^{13} states into about 10_4 or 10^5 descriptions, depending on the exact choice of attributes.

Although this compression is great, it is not the main benefit of the representation. The main benefit is to tame the evaluation function H. H maps states into probabilities, which estimate the likelihood that a state will appear on a shortest-solution path. This probability H varies with the attributes X_i *monotonically.* For example, the smaller the city-block distance, the more likely the state will be useful. Although attributes may interact somewhat, the monotonic relationship allows us to assume a linear combination: H(x) = b_iX_1 + b_2x_2 +— + b_k x_k = b.x, and we need only learn the weights b_i.

H can also be viewed as a class membership function, where the class is probabilistic (cf. Fig. lb). H is comparable to other concepts that begin with favorable representations. In a favorable representation the attributes are matched to the problem so that, over their instance space, concepts exhibit few disjuncts or peaks [Holte & Porter, 1988; Rendell, 1988]. If our linear model b.x is appropriate, then H has just one peak — where each x_i has its extreme value.

Associated with H is its *weight space,* over which is defined some *objective function* [Gill et al., 1981]. Our objective is task performance. Figure 2 shows that a performance function over a weight space defined by our attributes tends to be smooth, and may have a single optimum. The ordinate shows the average number of nodes developed before a solution was found, for a large set of arbitrarily difficult puzzles. Weight space is hypothesis space, the continuous analog of a discrete version space [Mitchell, 1978]. In version space, hypotheses are correct or incorrect. In weight space the hypotheses have degrees of correctness, shown in Figure 2 with the best at the central position of each graph. Such optima are surprisingly difficult to find, even for simpler spaces.

Although our representation gives typical membership and objective functions, this problem is harder than some because it requires dynamic learning. This produces biased data, which are already uncertain. The data come from search trees; each node becomes a training example. An example is positive if and only if it appears on a solution path. Although failures to solve give only negative examples, incomplete search trees are useful for training when combined with successful searches. Depending on the learning method, several searches may compose a single iteration. An iteration is a set of searches to gather many data, followed by the computation of an improved evaluation function H to guide new searches. Because H favors useful states, iterated learning requires "unbiasing" procedures [Rendell, 1983].

3. A Selection of Learning Methods

The weights of an evaluation function or the parameters of any concept description can be learned in many different ways. However, there are two *basic* approaches. One learns the function over instance space (Fig. 1); the other works with hypothesis space (Fig. 2). Methods that use instance space directly include curve fitting and decision tree induction. Methods that work in hypothesis space include candidate elimination and optimization techniques. We first outline, then evaluate three instance space methods, two hypothesis space methods, and one combination method (see Table I).

(1) Direct curve fitting. If attributes have real, integer, or binary scales, we can use statistical *regression* [Draper & Smith, 1981], which fits the best hyperplane $H(x) = b_1x_1 + b_2x_2 + \cdots + b_kx_k$ by minimizing the least squared-error for different choices of weights b_i. Abstractly, this is like searching weight space (Fig. 2) to minimize the objective function — here the squared-error. Operationally, however, regression is algorithmic — no search is necessary, because the technique inverts matrices to solve minimization equations directly. Hence the method is very fast.

If our class membership values were probabilities, normal regression techniques [Draper & Smith, 1981] would apply. However, in our experiments (and in many machine learning applications) the data are binary. Fortunately, binary data can be managed by analogous techniques, one of which is *probit* analysis [Finney, 1971].

(2) Probability regions and discrete evaluation. In machine learning the best known empirical technique may be induction of decision trees or partitioning of instance space [Quinlan, 1986]. For graded concepts, the instance space is divided into regions of similar probability [Breimah et al., 1984]. If we have learned a concept, these regions classify instances into their probable class; if we have learned an evaluation function, the regions classify instances into their probable utility [Rendell, 1983].

For our fifteen puzzle problem, one pass of a partitioning algorithm is insufficient. Because of the difficulty of this puzzle, early data must come from easier problems, and later data from intelligent search (causing sample bias). Hence, the partitioning algorithm becomes just one operation in a scheme to form and revise regions and their probability estimates [Rendell, 1983]. After the initial partitioning, later passes use the biased data and three other operations: *unbiasing, probability updating,* and *region refinement. Unbiasing* begins by comparing averaged data within existing regions, which already provide an unbiased estimate of the probabilities. These unbiased estimates are now compared with the biased data, pairwise for each region, to extract a relationship. The relationship allows an operation to be applied to the biased probabilities to convert them to unbiased estimates. Now the new probability estimates are averaged with the old to provide *probability updating.* The newly unbiased data are also used for a different operation: *region refinement,* which further subdivides the existing partition.

(3) Fitting probability regions for smoothed evaluation. Although the regions output by an induction algorithm allow refined classification, the discrete nature of this approach may be insufficient. As we see in Section 4, search performance may be poor if the discrete probability regions do not

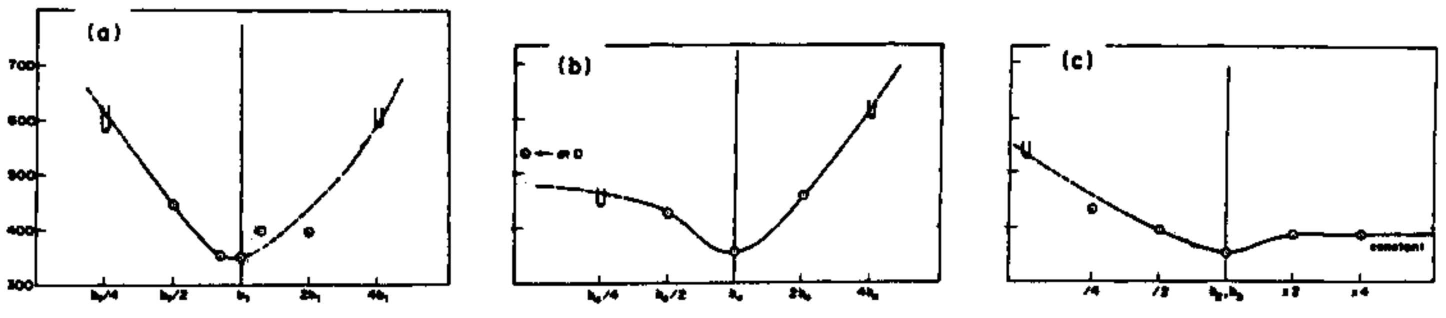

Figure 2. Fifteen puzzle search performance as a function of four weights in an evaluation function expressed as a linear combination of attributes. The performance measure μ is the average number of nodes developed before solution of 32 hard problems.

discriminate well enough. To address this problem, we could use the probability regions as data in statistical regression to find the best smoothed function $H(x) = b_1x_1 + b_2x_2$ -K...4- b_kx_k.

Instead of this complicated process of inducing regions, then fitting them, why not simply fit the original data? The answer is that the computational resources required to unbias are too great if we retain all the data. In contrast, if we iteratively repeat the operations of data gathering, unbiasing, partitioning, then regression, then our primary information structure is the compressed regions. Regions are easier to update.

(4) Optimisation using response surface fitting. Unlike the previous three methods which work in instance space (Fig. 1), optimization methods search hypothesis space to minimize (maximize) the objective function μ. Because the weights in Figure 2 have real values, and because μ is smooth and appears to have only one peak, a hill-climbing method is suitable [Gill et al, 1981]. A hill-climbing technique selects weight vectors b by moving in the direction of improving μ(b). One technique suitable for noisy domains is response surface fitting with a diminishing grid. In this method we select points b at corners and midpoints of a hypercube in weight space, then gather data (μ values) to fit a quadratic (a parabola). The optimum predicted by the parabola allows us to gather more refined data, as we gradually hone in on the optimum μ by repeatedly shrinking the grid. Starting with a large grid detects broad tendencies; shrinking the grid improves accuracy.

Despite the quality of this method, a serious problem in our case is that most values of μ cannot be found! This is because most choices of b give such poor performance that typical problems cannot be solved within reasonable time. To offset this problem, initial runs used easier problems.

(5) Optimization using a genetic algorithm. Genetic algorithms are designed to optimize an objective function μ called the fitness [Holland, 1975]. Given a population, hypotheses b are selected stochastically for breeding, with probabilities proportionate to μ(b). Hypotheses are usually represented as bit vectors called genotypes. Binary operations such as crossover are applied to pairs of genotypes; unary operations such as mutation are applied to single genotypes. The operations produce a new generation of hypotheses. Because they implement parallel search, genetic algorithms can manage badly behaved objective functions.

For this reason, a genetic algorithm seems unnecessarily powerful for the problem illustrated in Figure 2. Massively parallel search is not required and it may be costly. We still have the problem of computing μ for mediocre choices of b, though once again we can do some preliminary search.

(6) Combining partitioning and parallelism. Our final method is a combination of partitioning in instance space and parallel search in hypothesis space. This method uses a modified genetic algorithm to govern multiple partitioning [Rendell, 1985]. The "genotype" is compressed and variable — it is a set of probability regions (Fig. lb). Each of these structures produces a different evaluation function H, which is then given some puzzles to produce data of two types: detailed data for updating probabilities and refining regions, and overall data for measuring performance μ. This allows the selection of individual regions for k-sexual crossover.

4. Comparative Results

Table I summarizes representative results of many experiments for each of the six methods described in the previous section. To assess these methods we use the number of nodes developed μ. Because other computation is negligible, μ is a good measure of both the learning time and the quality of the resulting evaluation function. (Another measure, the length of the solution, was found to track μ.) On a VAX 780, 1000 nodes take about 1/2 minute CPU time. The average values of μ shown in the table were obtained by solving 100 puzzles, giving a standard error of about 11 (or 3%). Although some related work appeared in [Rendell, 1983] and [Rendell, 1985], most of these results are new. All experiments used the same four attributes of Figure 2.

(1) Direct curve fitting. To fit the best hyperplane $H(x) = b_1 X_1 + b_2x_2 + \ldots\ldots + b_kx_k$ to the binary data from search trees, probit analysis [Finney, 1971] was used. Because the technique is algorithmic, it is fast. However, the quality (nodes developed) is 534— 53% worse than optimal, and this is after favorable interpretation of the results. To begin the experiment, puzzles nine moves from the goal were given (greater difficulty requires too much computation). The resulting data gave a non-zero weight only for the city-block distance x_1 because easy puzzles can hardly have the impediments described by the other attributes x_2, x_3, and x_4. To continue the experiment, the resulting evaluation function was used to solve harder puzzles and find impediment weights. These weights (b_2, b_3, and b_4), now approximated the correct ones, but because of the biasing effect of the city-block distance already in H, the new value of b_1 was in error by 1200%. Further experiment gave similarly biased and unpredictable values, although if we take the value of b_1 from the first iteration and the values of b_2, b_3, and b_4 from the second, we obtain the performance shown.

(2) Probability regions and discrete evaluation. The second method is to partition instance space into regions of similar probability (of a state's appearing on a short solution). Because the technique is iterative, it requires not only initial partitioning, but also partition refinement after data unbiasing (see Section 3). The improving quality of the evaluation function over repeated iterations allows the solution of harder problems, which provide more representative data. This alternation of sampling and learning allows each process to speed the other. But the evaluation is discrete:

For task performance the states are classified into discrete regions. This lack of smoothing or interpolation causes poor performance: at least four times optimal quality. (The number of nodes developed could not be tested precisely, because resource limits were often exceeded.)

(8) Fitting probability regions for smoothed evaluation. This is the same as the previous method except that at the end of each iteration the probability regions are used as data to find weights b_i for the linear combination $H(x) = b_1x_i + b_2x_2 + \_\_\_ + b_kx_k$. After convergence in half a dozen iterations, this smoothed evaluation function gave near-optimal performance of 353 (compare Fig. 2 and see [Rendell, 1983]). Although H contains the only knowledge used for solving, the probability regions provide the primary information, and are more suitable for dynamic learning. One advantage of this and other instance space methods is that *every state counts.* One drawback of this and most methods is that for fast learning, the difficulty of the training problems must be just at the current performance limit. This requires some user experience. Moreover, the combination of approximate techniques (e.g., unbiasing, and search that relies on previous learning) over repeated iterations can cause problems. To some extent the iterative learning seems to be self-correcting, but often performance degrades slightly or levels off. One cure is user experience; another is repeated runs. Experiments have shown that about ten runs are required for a result within 10% of optimal. A similar criticism applies to most methods, so their learning times are multiplied by ten in cololmn 5 of Table I ("Effective Cost").

(4) Optimisation using response surface fitting. Rather than probabilities over instance space, optimization methods use summary measures of performance μ over hypothesis space. Summary measures cost more to obtain: concept accuracy requires the classification of many instances; search performance requires the solution of whole problems. For each problem solved, only a single value is obtained — the number of states developed. This contrasts with the first three methods, which identify each count with a point in instance space. This design difference explains the difference in learning times: hill climbing in weight space is slower. Initially, most values of the weight vector b are so poor that problems cannot be solved. To counteract this problem, a preliminary round of curve fitting (row I) was used to obtain approximately correct weights.

(5) Optimization using a genetic algorithm. Approximately correct weights were also given to the genetic algorithm (row 5). Genetic algorithms need a well-chosen representation. If the genotype is too short, resolution will be lost; if this bit string is too long, time will be wasted. To ensure better performance, the graphs in Figure 2 and some preliminary runs were analyzed to choose a genotype length of six bits. Another variable is the population size. Several experiments used populations up to 200. The less than optimal performance of 388 (12% worse than optimal) for the best individual in a population of 50 is perhaps not too surprising because genetic algorithms are not designed for obtaining 100% accuracy when the objective function μ is unimodal, but rather for approaching multiple optima in parallel when the function is badly behaved. The high cost results from so much search (hundreds of states in each of many problems) for so little (a single performance value for each solution). Another problem is that verification of an individual weight vector requires a larger sample than during learning. For each weight vector suspected to be close to optimal, many test problems must be solved, which typically costs 10,000 nodes per candidate vector.

(6) Combining partitioning and parallelism. Perhaps surprising is the sixth result. This gives the best performance (although not significantly better than row 3 because the standard error is about 11, or 3%, in all rows). The cost appears higher than for the extended partitioning method, by a factor of six. Superficially, then, it seems that extended partitioning

Table I. Evaluation of Six Learning Methods

Learning Method	Concept Quality *	Optimality Factor	Learning Cost *	Effective Cost	Comments
Curve Fitting (with linear model)	534	1.53	11,000	~110,000	Performance as good as shown needs human help.
Iterative Partitioning and Updating	>1500	>4	28,000	~280,000	Performance is poor because of discrete probability classes.
Iterative Partitioning, Updating, and Regression	353	1.01	17,000	~170,000	Learning this quickly needs knowledge about training.
Response Surface Fitting (optimization)	421 381	1.21 1.09	72,000 345,000	~720,000 ~3,450,000	Cost much greater unless optimum approximately known at start.
Genetic Algorithm (optimization)	388	1.11	280,000	~2,800,000	Taking best individual of various populations over generations.
Partitioning, Regression, and Genetic Algorithm	348	1	89,000	89,000	This combination method is the only one without hidden costs.

* Measured as number of nodes developed to solve or learn (see text).

is better when used alone than when combined with a genetic algorithm. However, this cursory assessment is misleading. The favorable learning speed of extended partitioning in row 3 results from considerable user experience with training. In fact the learning speeds given in rows 2 through 5 are about an order of magnitude too low (reflected in col. 5). In contrast, the combined method of row 8 needs little user guidance because it is much less sensitive to training problems and evaluation errors. Even with a small population of 10 or 20, this method is extremely *stable.* It is reliable and easy to use. Furthermore, this method avoids the cost of verification, because individual weight vectors need not be tested. Rather, *all* the regions from *all* the individual partitions can be used as a single large data set to fit a very accurate evaluation function ([Rendell, 1985] elaborates).

5. Discussion

This study of six empirical-learning methods suggests ways to cope with domains that are simultaneously probabilistic, dynamic, noisy, and biased. One recommendation is to combine types of methods. Even for our numeric domain, the standard methods of curve fitting (1) and optimization (4) were limited because of biased data (in l) and lost information (in 4). Furthermore, a standard method of instance space partitioning or decision tree induction, despite its extension for progressive refinement after data "unbiasing" (2), was inadequate because discrete classification rules were too unrefined, even though they were probabilistic. But when partitioning (2) was combined with curve fitting (1), the learning was fast and the task performance was optimal. Incorporating a third technique improved learning behavior still more. Like method 4, the genetic algorithm (5) could use only some of the information available in the search domain, and although this method (5) behaved relatively poorly when used alone, it stabilized method 3 and made it easier to use (cf. [Quinlan, 1988]).

Because other problems exhibit characteristics similar to the search problem we analyzed, the phenomena should generalize. Probabilistic evaluation functions are probabilistic concepts (Section 2.1). In many real-world domains the concept is probabilistic, the learning situation is dynamic, and the data are noisy and even biased (Section 2.2). To reiterate the phenomena we observed:

- Instance space algorithms find class membership values H as a function of attribute values x; hypothesis space algorithms optimize overall quality values μ of entire functions H(x). Combinations of these two methods exploit both kinds of information.
- Instance space algorithms are fast; hypothesis space algorithms are stable. Combinations may have both advantages and also be easier to use.

The varied strengths of different techniques may provide a net gain when the methods are combined [Ackley, 1985], Ultimately, systems for very general learning may owe much to a structured combination of techniques [Buchanan et al., 1978].

References

Ackley, D. H. A Connectionist Algorithm for Genetic Search. *Proc. International Conference on Genetic Algorithms and their Applications,* 1985, 121-135.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. *Classification and Regression Trees.* Belmont, California: Wads-worth, 1984.

Buchanan, B. G., Johnson, C. R., Mitchell, T. M., & Smith, R. G. Models of Learning Systems. In J. Belzer (Ed.), *Encyclopedia of Computer Science and Technology.* 1978.

Doran, J., & Michie, D. Experiments with the Graph-Traverser Program. *Proc. Roy. Soc,* 1966, *A„* 235-259.

Draper, N. R., & Smith, H. *Applied Regression Analysis.* Wiley, 1981.

Finney, D. J.. *Probit Analysis.* Cambridge University Press, 1971.

Gill, P. E., Murray, W., & Wright, M. H. *Practical Optimization.* New York: Academic Press, 1981.

Holland, J. H. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

Holte, R. C, & Porter, B. W. *An Empirical Study of Bias Appropriateness.* Austin, Texas, 1988.

Langley, P. A General Theory of Discrimination Learning. In David Klahr (Ed.), *Production System Models of Learning and Development.* Cambridge, MA: MIT Press, 1987.

Mitchell, T. M. *Version Spaces: An Approach to Concept Learning.* Stanford Ph.D. Thesis, 1978.

Quinlan, J. R. The Effect of Noise on Concept Learning. In R. S. Michalski (Ed.), *Machine Learning: An Artificial Intelligence Approach.* Kaufman, 1986.

Quinlan, J. R. *An Empirical Comparison of Genetic and Decision-Tree Classifiers.* Proceedings of the Fifth International Conference on Machine Learning, Ann Arbor, Michigan, June 12-14, 1988.

Rendell, L. A. *An Adaptive Plan for State-Space Problems.* Dept of Computer Science CS-81-13 University of Waterloo Ph.D. Thesis, 1981.

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

Rendell, L. A. Genetic Plans and the Probabilistic Learning System: Synthesis and Results. *Proc. International Conference on Genetic Algorithms and their Applications,* 1985, 60-73.

Rendell, L. A. Learning Hard Concepts. *Proceedings of the Third European Working Session on Learning,* 1988, 177-200.

Samuel, A. L. Some Studies in Machine Learning Using the Game of Checkers. In E. A. Feigenbaum (Ed.), *Computers and Thought.* McGraw-Hill, 1963.

Winston, P. H. *Artificial Intelligence.* Addison Wesley, 1984.