Machine Learning 2: 199 228, 1987
© 1987 Kluwer Academic Publishers, Boston · Manufactured in The Netherlands

Classifier Systems and the Animat Problem

STEWART W. WILSON (WILSON@THINK.COM)
The Rowland Institute for Science, 100 Cambridge Parkway, Cambridge, Massachusetts 02142, U.S.A.

(Received: February 15, 1986)
(Revised: February 26, 1987)

Keywords: Classifier systems, incremental learning, disjunctive concepts, payoff, animal learning, genetic algorithm

Abstract. This paper characterizes and investigates, from the perspective of machine learning and, particularly, classifier systems, the learning problem faced by animals and autonomous robots (here collectively termed *animats*). We suggest that, to survive in their environments, animats must in effect learn multiple disjunctive concepts incrementally under payoff (needs-satisfying) feedback. A review of machine learning techniques indicates that most relax at least one of these constraints. In theory, classifier systems satisfy the constraints, but tests have been limited. We show how the standard classifier system model applies to the animat learning problem. Then, in the experimental part of the paper, we specialize the model and test it in a problem environment satisfying the constraints and consisting of a difficult, disjunctive Boolean function drawn from the machine learning literature. Results include: learning the function in significantly fewer trials than a neural-network method; learning under payoff regimes that include both noisy payoff and partial reward for suboptimal performance; demonstration, in a classifier system, of a theoretically predicted property of genetic algorithms: the superiority of crossovers to point mutations; and automatic control of variation (search) rate based on system entropy. We conclude that the results support the classifier system approach to the animat problem, but suggest work aimed at the emergence of behavioral hierarchies of classifiers to offset slower learning rates in larger problems.

1. Introduction

This paper characterizes an important but quite unexplored machine learning problem, assesses the relevance of existing techniques, and focuses on a limited but challenging test of one technique - *classifier systems* (Holland, 1976, 1986) - that appears particularly appropriate to the problem.

1.1 The Animat problem

To survive in its environment, an animal must possess associations between environmental signals and actions that will lead to satisfaction of its needs. The animal is born with some associations, but the rest must be learned through experience. A similar situation might be said to hold for an autonomous robot (say on Mars or under the sea). One general way to represent the associations is by condition-action rules in which the conditions match aspects of the animal's environment and internal state and the actions modify the internal state or execute motor commands. We are concerned with how, from a computational standpoint, the animal could use its experience to discover an effective (i.e., needs-serving) set of rules.

This learning problem is difficult because information is hard to obtain. In the first place, because the animal must keep on about its business, instances of environmental situations relevant to development of particular rules come one at a time, in arbitrary order, interspersed with instances relevant to other rules. Though in principle large numbers of instances might be stored over time and processed at leisure, it seems more realistic to assume that the animal has only limited storage capacity for raw experience and must extract information from each instance when it occurs. We are thus assuming that the animal learns *incrementally*, with essentially no direct memory of events in their original form or detail. (Past events are of course implicitly represented – more or less well – by the evolving rules, and the power to "recall" an event – again more or less well – by rule manipulation is not excluded by the incremental learning constraint.)

Second, the animal does not in general have the luxury of a teacher-like environment to tell it which action was "right" (most needs-serving) in each situation, or even to provide an error signal. Instead, on those relatively rare occasions when an action brings immediate satisfaction of a current need, the best the animal can do is rate the action (and the responsible rule) according to the degree of satisfaction. However, there is no assurance that some *other* action would not have resulted in greater satisfaction. We shall use the term *payoff* as a shorthand for "degree of need satisfaction," and shall speak of the animal's "receiving payoff" from the environment but will always mean by this that the animal is experiencing some degree of need satisfaction as a consequence of being in a certain relationship to the environment.

The payoff constraint is actually even tougher than this because usually, after an action, no payoff at all is received. Yet that action (e.g., a certain move) may have located the animal so that the *next* action will result in a large payoff. Such "stage-setting" action chains can be long. To support them, the animal must somehow get proper credit to the constituent steps, but in a way that does not exceed the animal's limited raw storage capacity. Thus in general the animal has to learn under payoff that may be *delayed.*

Finally, though the animal may have some built-in preprocessing of sensory signals, useful combinations of perceptual elements are not given in general and must be created by the animal. To reflect this, we assume that the environmental signals are vectors of low-level features taking on discrete values. In learning, the animal must discover significant combinations of the features, generalizing them as much as possible (while still avoiding error) to cope with the great diversity of the environment. Irrelevant features, depending on the situation, should come to be ignored. From a logical standpoint, the resulting concepts will usually be *disjunctive*: for example, the many reasons why the animal might "move left" will tend to have little in common.

Our underlying hypothesis, and the motivation for the more specialized work in the experimental part of the paper, is that study by computer simulation of learning under the above conditions should be useful both for understanding learning in organisms and for the design of highly adaptive, autonomous robots. We have termed such simulations *artificial animals* or, briefly, *animats* (Wilson, 1985); we also apply the term *animat* informally to the animals and autonomous robots themselves. A simulation requires specification of environment, needs, sensory and motor equipment, and learning method.

In light of the constraints we have assumed above, the "animat problem" may be characterized technically as: incremental learning of multiple disjunctive concepts under payoff. To the extent this is an accurate characterization of the *organism's* learning problem, simulations that succeed in emulating an organism's behavior in realistic environments should, in their computational principles, suggest models of the organisms themselves, which can be compared with models from other sources (e.g., experimental psychology, neurophysiology). Similarly, the animat approach should lead to design principles for autonomous robots, though here one might suggest that the constraints are too strict. For example, since computer memory is inexpensive, why require incremental learning when large amounts of raw experience (for "training sets") could presumably be stored? One answer is that if natural organisms *do* learn incrementally, then since organisms are remarkably competent, robot designs should benefit from understanding how organisms manage *without* storing raw experience.

1.2 Available approaches

Many elegant techniques of machine learning turn out to be inapplicable to the animat problem because the techniques require more information than the problem constraints permit. Michalski (1986) distinguishes three major research paradigms in machine learning: neural modeling and decision theoretic techniques; symbolic concept acquisition; and knowledge-intensive, domain-specific learning. The last covers systems having significant amounts of built-in domain knowledge. The animat, though it may have some instinctive responses, does not in general have prior knowledge of the world, so that the third paradigm is not directly of interest.

Within the symbolic concept acquisition paradigm, the programs of Winston (1975) and Mitchell (1982) are prototypes for incremental learning of single concepts from examples, but they have trouble handling disjunctive concepts and noise (also important in the animat world), and require a teacher-like environment that labels the examples. Michalski and Larson's (1978) AQ^{11} program and Quinlan's (1983) ID3 program learn multiple, disjunctive concepts from (potentially noisy) examples but, because they also expect the examples to be pre-classified, they are not capable of learning under payoff. In addition, ID3 is not incremental. AQ^{11} is described as incremental by its authors, but there appear to be no published reports of tests under the incremental constraint of the animat problem: one example at a time with no storage of past events.[1]

Within Michalski's first paradigm, neural modeling and decision theoretic techniques, two approaches using networks are representative. Both do learn disjunctive concepts incrementally. The associations that the animat needs to learn can be regarded as mappings from input strings to output strings. Rumelhart, Hinton, and Williams (1985) discovered a generalization of the perceptron learning rule that may permit a network of nonlinear, neuron-like elements to learn an arbitrary binary mapping. From the animat point of view, however, the technique has the drawback that learning requires the environment to supply the correct value for each bit of the output string. The animat, on the other hand, receives only payoff (i.e., a single scalar) for its output action as a whole. Even if there are only two action possibilities, the correct choice is not necessarily indicated: the payoff may be a quantity like "37."

The other network approach (Barto, 1985) is noteworthy because the algorithm for adjusting the input weights of its so-called A_{R-P} elements is partly stochastic. This is believed to cause the network to explore more thoroughly the space of possible activity patterns, which indirectly causes the learning algorithm to explore more thoroughly the weight space. Such a partly stochastic search can be helpful in avoiding entrapment at suboptimal solutions (Kirkpatrick, Gelatt, & Vecchi, 1983). Like Rumelhart et al., Barto presents experiments showing his network learning several difficult nonlinear mappings. However, all are from an input string to $\{0, 1\}$, leaving open the question of how the technique would apply to string-string mappings. More broadly, from the animat viewpoint, it is not clear how

[1] Anderson's (1983) ACT* system, which its author describes as a "theory of cognitive architecture," is potentially quite relevant to the animat problem, but we defer its discussion until section 2.4.

feedback in the form of payoff could be used.

The *classifier systems* originated by Holland (1976, 1986) are included in Michalski's first paradigm, and are the focus of this paper. Classifier systems appear to offer a broadly useful framework for addressing the animat problem. In theory, a classifier system is suited to learn multiple disjunctive concepts incrementally under payoff (including noisy payoff and delayed payoff in sequential tasks). But the method has been investigated experimentally in only a few cases. Holland and Reitman (1978) exhibited successful performance by an animat-like classifier system that optimized its rate of satisfaction of two distinct needs. Booker (1982) experimented with an animat-like "hypothetical organism" in a simple environment that contained both attractive and aversive stimuli. Goldberg (1983) used a classifier system to adaptively control a simulated Newtonian system and a gas pipeline. While the results in each case showed the potential of classifier systems, the problems solved were from a logical standpoint fairly small.[2]

Wilson (1985) used a classifier system for an animat that lived in an environment containing 92 distinct sensory vectors. The animat's objective was to satisfy its need for "food." With no initial knowledge of what food looked like, the animat evolved classifiers (rules) that led it not only to move toward food when it was in view but also to move on efficient multi-step paths toward food not in immediate view, using other objects as clues to the proximity of food. The system formed eight disjunctive generalizations, showing the potential of classifier systems for larger problems.

These modest demonstrations suggest further investigation of the approach's power. For example, how logically difficult a problem can be handled, and on what learning time scale? What determines the degree of generality achieved by the classifiers, and can this be controlled? What is the effect of different payoff regimes, including the use of penalties? One way to investigate these questions is through additional simulations of the kind cited. Another way is through focusing on a single, well-defined, but difficult task that would challenge the method's basic mechanisms. In the present research, we chose the latter course. We decided for simplicity to leave out delayed payoff and concentrate on a non-sequential problem environment in which the core animat constraints - incremental learning of disjuncts under (immediate) payoff - would still hold. As a test task, we chose the Boolean multiplexer, a difficult logical function that would

[2] Smith (1980) employed a significant variant of the classifier system idea in a poker-learning system that rivaled Waterman's (1970) results. His interesting approach, which applied the genetic algorithm to a population of classifier systems instead of a population of classifiers, falls substantially outside the present framework, and we do not cover it here. Also see Schaffer (1985).

permit comparison with Barto's results on the same task. In the next section we review the standard classifier system model and then, in Section 3, describe ***Boole***, our specialization of the standard system for Boolean problems. In Section 4 we present experiments on the questions above.

2. The standard classifier system

We shall refer to the system described by Holland (1986) as the "standard" classifier system.[3] Structurally, it consists of (1) a finite population [P] of fixed-length condition-action rules called *classifiers*, (2) a message list, (3) an input interface consisting of a set of environmental feature detectors, and (4) an output interface for affecting the environment. Sources of environmental payoff must also be defined. Functionally, the system employs three algorithms: a performance algorithm, a reinforcement algorithm (called the *bucket brigade*), and a discovery algorithm.

The classifier population may be thought of as a set of hypotheses representing the system's current estimate of the best means of obtaining payoff. The performance algorithm, together with the classifier population, determines the system's short-term behavior. The reinforcement algorithm defines how received payoffs alter classifier strengths; that is, how credit is to be apportioned to the apparently responsible classifiers. The discovery algorithm uses a version of Holland's (1975) genetic algorithm to generate new, possibly better classifiers by recombining "building blocks" from existing high-strength classifiers and inserting the new classifiers into the population to compete for payoff.

2.1 Representation: Classifiers and messages

The system's basic elements are *classifiers* and *messages*. All messages are strings of length L from $\{1,0\}$. All classifiers have the structure:

$$t_1, t_2, \ldots t_r/a \qquad ,$$

where the condition and action are separated by "/". The condition consists of r individual *taxa*, t_i; the action, a, is a message. Each taxon is a string of length L from $\{1, 0, \#\}$. The condition is satisfied if and only if every t_i matches some message currently on the message list. An individual t_i *matches* a message if and only if for every 1 or 0 in t_i, the same value occurs at the corresponding position in the message; "#" functions as a "don't care" symbol in a taxon and matches unconditionally. The #'s confer generality on the taxon (and thus on the classifier) in the sense that

[3] Our summary will omit some details of the standard approach.

2^n distinct messages can be matched by a taxon containing n #'s. As just mentioned, each classifier also possesses a *strength*, a scalar quantity that is adjusted by the reinforcement algorithm and that represents an estimate of the classifier's value to the system.

Messages from the environmental interface have the same format as classifier action messages, i.e., they are strings of length L from $\{1,0\}$. The output interface is a pre-defined mapping from some subset of the message space to actual actions in the environment. For example, the interface might respond to a message ending in "0110" by causing the system to take a step forward, etc.

2.2 Performance and reinforcement algorithms

The operation of the performance and reinforcement algorithms consists of an iterated *major cycle* in which: (1) messages from the input interface are added to the message list; then (2) classifiers whose conditions are satisfied by messages on the list post their own messages to the list (and the old messages are erased); and finally (3) any posted messages that would trigger the output interface do so, but with resolution of effector conflicts if necessary. In the *bucket-brigade* reinforcement algorithm, a (small) fraction e of the strength of each posting classifier is transferred to the strengths of the classifiers that left the messages matched by the posting classifier (with the recipient classifiers sharing the transferred amount equally in the simplest case). When external payoff enters the system, it is shared by the classifiers whose messages are currently on the list.

A classifier whose message is matched by a classifier on the next cycle is said to be "coupled" to that classifier. If a sequence of coupled classifiers leads to external payoff and the sequence gets repeated, strength increments will in effect flow back, via the bucket brigade, to reinforce the sequence's early-acting members. Thus early-acting, "stage-setting" classifiers that make possible later payoff will receive due credit, even though the payoff is delayed. In general, classifiers that match on a cycle do not automatically have the right to post their messages, but must have strengths above a threshold. Apart from this requirement, all matching classifiers gain the opportunity to compete for the bucket brigade's payoff flow.

2.3 Discovery component

In the performance and reinforcement algorithms, classifiers that lead to more payoff increase in strength and thus increase their own, and the system's, effectiveness in gaining payoff. In the discovery component, strengths play a distinct further role, influencing the generation of *new* classifiers that are likely to be better at getting payoff than existing ones.

The discovery component, using a version of Holland's (1975) genetic algorithm, can be described as a multiple search through the space of classifier structures along trajectories influenced by the structures of current above-average (strength) classifiers. Holland presents a deeper perspective in viewing the current classifiers as sampling an underlying space of classifier "building blocks" or substrings (including discontinuous substrings). He proves that the mechanisms of the genetic algorithm efficiently test and rank extant substrings while continually introducing new substrings and biasing the search of substring space toward above-average regions. Improvement in substring space leads to improvement in classifier space (and therefore of the system) since, under the algorithm, instances of the best substrings tend to be combined to form new classifiers.

Despite the subtle action just suggested, the basic cycle of the discovery component is quite simple. Classifiers ("parents") are selected probabilistically from the population [P] according to strength and then copied. Genetic operators are applied to at least some of the copies. Then all of the copies ("offspring") are added to [P], replacing a like number of weak classifiers. The principal genetic operators are *crossover* and *mutation.* In crossover, a randomly selected segment is exchanged between a pair of classifiers. In mutation, the values (alleles) at one or more positions (loci) in a classifier are changed to one of the other possible values; e.g., 1 would be changed to either 0 or # in the case of a taxon allele.

The discovery component operates in the background, producing offspring at a certain rate and inserting them in [P]. The offspring compete with the parents, as well as with other members of [P] that may match in similar situations. If the new classifiers are more effective in gaining payoff than their competitors, they will survive and the competitors may eventually be replaced. The new classifiers will not survive if they do not better their competitors in a sufficiently large set of situations.

2.4 Comparison with ACT*

Classifier systems are related to production systems (Davis & King, 1977) in that both employ condition-action rules, message-passing, and a recognize-act cycle. Holland (1986) discusses the differences in detail. Anderson's (1983) system ACT* is perhaps the most sophisticated *learning* production system (but see also Laird, Rosenbloom, & Newell, 1986). ACT* includes strengthening mechanisms for productions as well as generalization, discrimination, and other operators. The system was motivated by, and designed to simulate, results from human experimental psychology.

Perhaps the largest differences with classifier systems concern representation and discovery. ACT* productions manipulate "high-level" concepts (e.g., "red triangle," "Baptist"); the perceptual machinery for generating such symbols in the first place tends to be assumed. In contrast, classifier systems propose to *create* all required symbols - perceptual to high-level through system action upon binary strings. Second, ACT*'s discovery operators tend to be applied for reasons, or *purposefully* (e.g., discrimination may be triggered by observed correct and incorrect applications of a production). In contrast, classifier system operators act *randomly* on copies of successful (strong) classifiers.[4] ACT* appears to satisfy the incremental and disjunctive animat constraints (Anderson & Kline, 1979), though its ability to learn under payoff is less clear. The two systems should be compared on similar problems, if possible.

3. ***Boole***: A specialized classifier system

The above discussion of the standard classifier system introduced broad principles of operation. In this section we describe ***Boole***, a specialized classifier system for Boolean functions that we used for experimenting with non-sequential learning under the core animat constraints. A Boolean function of L variables is a mapping from binary strings of length L to $\{0, 1\}$. Learning the function means acquiring the ability to give the correct output value, 0 or 1, for any input string. A Boolean function makes a good task since a difficult, disjunctive function (concept) is easy to define and instances of known value are readily generated. Because all needed information is available in the input string, ***Boole*** omits the standard system's sequential aspects – the message list and bucket brigade - and makes its decision in a single time step.

3.1 Representation

Boole's classifiers each consist of a single taxon and an action. For L-bit input strings, the taxon has length L. The action is simply a single bit, 0 or 1, representing one of the two possible values for the function. The classifier says, in effect, "if my taxon matches the current input string, consider making my action the system's decision." An example of a classifier for $L = 6$ would be:

$$1 \ 0 \ \# \ 0 \ 1 \ 0 \ / \ 1 \ .$$

The classifier population [P] is usually initialized by filling all taxa with 0, 1, and # according to some random rule; actions are similarly filled

[4] The distinction is somewhat reminiscent of that between Lamarckian and Darwinian evolutionary mechanisms (Curtis, 1983).

in. Except as specifically noted, in all the experiments reported here, [P] contained 400 classifiers and was initialized with a uniform random distribution. Initial strengths were set to a common value (100).

3.2 Performance component

In the performance cycle an input string is presented and the system decides on its "answer": 0 or 1. The cycle has just two steps:

1. Form the *match set* [M] of all classifiers whose taxa match the input string.
2. Select a single classifier from [M] using a probability distribution over the strengths of [M]'s classifiers; that is, the probability of selection of a particular classifier is equal to its strength divided by the sum of the strengths of classifiers in [M]. Output the selected classifier's action as the system's decision.

In effect, the system asks which classifiers in [P] "recognize" the current input, then from these tends to choose the action having the greatest total strength among the classifiers advocating that action.

3.3 Reinforcement component

The reinforcement component adjusts classifier strengths in accordance with payoff received from the environment following each performance cycle. There are a variety of possible schemes, and the differences between them are an important focus of research. The schemes we have tried are encompassed by the following algorithm.

1. Form the subset of [M] consisting of classifiers whose action is the same as the chosen action. We call this the *action set* [A]. The remaining members of [M] are the set NOT[A].
2. Deduct a fraction e from the strengths of all classifiers in [A].
3. If the system's decision was correct, distribute a payoff quantity R to the strengths of [A]; but
4. If the decision was wrong, distribute a payoff quantity R' (where $0 \leq R' < R$) to the strengths of [A] and deduct a fraction p from the strengths of [A] (at least one of R' and p is equal to 0).
5. Deduct a fraction t from the strengths of NOT[A].

Particular reinforcement regimes will be discussed in detail later, but in general every [A] both pays out and receives. For a correct decision, the effect of reinforcement on [A] can be written:

$$S'_A = S_A - eS_A + R \quad , \tag{1}$$

where S_A is [A]'s total strength prior to payoff and S'_A is its total following payoff. Assuming that R is the same over time for this [A] (the simplest case), it can be shown that S_A will gradually approach R/e. This suggests that under reasonably steady payoff conditions, S_A will function as an *estimator* of typical payoff. Similarly, the strength of any individual classifier will be an estimator of *its* typical payoff. In general, equation (1) causes the strength to be a recency-weighted average of previous payoffs (divided by e); recent events get additional weight and the average changes faster for larger values of e.

The total payoff to [A] is distributed by a distribution function D, which in the simplest case gives each classifier an equal share of R. Biased distribution functions are important and will be discussed later. In any event, the more classifiers there are in [A], the less payoff each receives.

3.4 Discovery component

Boole's version of the genetic algorithm is as follows:

1. Select a first classifier C_1 of [P] using a probability distribution over the strengths of all classifiers in [P];
2. With probability χ, select a second classifier C_2 using the technique of step 1 and apply the crossover operation to copies of C_2 and C_1; choose one of the resulting pair as an offspring and discard the other;
3. If step 2 was not carried out, form an offspring by making a copy of C_1;
4. Apply the mutation operation to the offspring, changing each taxon allele with probability μ (choose the new allele equiprobably from the two possibilities);
5. If the offspring resulted from crossover, reduce each of the parents' strengths by one-third and set the offspring's initial strength to the sum of the reductions; else reduce C_1's strength by one-half and set the offspring's initial strength to the amount of the reduction (the total strength of parents and offspring is unchanged);
6. Add the offspring to [P];
7. Delete a classifier using a probability distribution over the inverses of the strengths of all classifiers in [P].

The algorithm is invoked with probability ρ per trial, i.e., per performance cycle. It is essentially the same as the standard system's algorithm, except that it produces just one offspring per invocation whereas the standard usually produces numerous offspring, but at longer time intervals. Our intention was to alter the population less discontinuously than the standard algorithm. Also, in the present algorithm, crossover and mutation actually applied only to the taxa; the action bit was not involved.

The discovery component has one further mechanism, called *creation*, which generates a new classifier during the performance cycle if [M] is empty. The taxon of the created classifier is a copy of the current input string, except that #'s are inserted with a probability equal to the current percentage of #'s in [P]. The action is chosen randomly,[5] and the strength is set to the mean strength of [P]. To make room in [P], a classifier is deleted as in step 7 above. Then [M] is recomputed and the performance cycle continues. Creation is an "emergency" measure that permits the system to extract information from, and respond to, unrecognized events, and prevents it from ignoring such events. Creation plays a minor role if the initial percentage of #'s is large enough, which was the case in the present experiments.

4. Experiments with ***Boole***

In this section we present four sets of experiments with the specialized classifier system, ***Boole***. The first subsection defines the class of Boolean functions that formed the problem environment.

4.1 The multiplexer problem

The Boolean "multiplexer" is a useful test function because it represents a rather intricate disjunctive concept, and generates a family of problems of increasing difficulty. In fact, for each integer $k > 0$ there is a multiplexer function of binary strings of length $L = k + 2^k$. The basic function can be defined by thinking of each input string as having k "address" bits a_i and 2^k "data" bits d_i, with the string represented by

$$a_0 a_1 \ldots a_{k-1} d_0 d_1 \ldots d_{2^k-1}$$

The value of the function is given by the value (0 or 1) of the data bit that is indexed by the pattern on the address bits. For example, the following

[5] Since the system is in a teacherless payoff environment, it has no immediate way to determine the correct action.

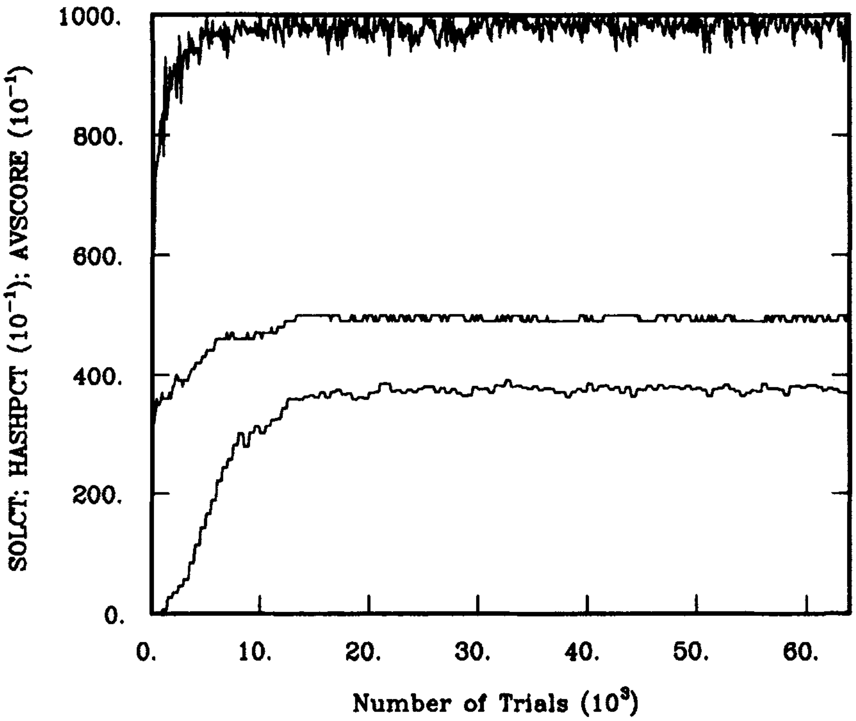

Figure 1. System quantities during solution of the 6-multiplexer. The upper curve represents a moving average of percentage correct decisions over past 50 trials (divide vertical scale by 10). The lower curve is the solution count, or the total number of instances of "solution set" [S6] in [P]. System parameters for the experiment were: $e = 0.1$, $\chi = 0.12$, $\mu = 0.001$, $G = 4.0$, $R' = 0$, $p = 0.8$, $\rho = 1.0$, $t = 0.1$.

are input strings and correct output values for the "6-multiplexer" ($L = 6$, $k = 2$) function:

0	0	0	1	0	1	0
1	1	0	0	0	1	1
1	0	1	1	0	1	0

Since $k = 2$, the address bits of the first example are 00, telling us to select as output the value of data bit 0, which is 0. In the second example the address bits are 11, so we look at data bit 3, with value 1, etc. Written in disjunctive normal form, the function is:

$$F_6 = a'_0a'_1d_0 + a'_0a_1d_1 + a_0a'_1d_2 + a_0a_1d_3 \quad ,$$

showing, since there are four terms, that the function is indeed disjunctive (in general there are 2^k terms).

We ran experiments with the 6-multiplexer, the 11-multiplexer ($L = 11$, $k = 3$), and the 20-multiplexer ($k = 4$). In each case ***Boole***'s task was to learn to give the correct value of the function for any input string. Strings were generated by a uniform random process and presented one at a time (i.e., incrementally), with each presentation constituting a *trial.* After each trial ***Boole*** received payoff according to its decision and the particular payoff regime in effect.

4.2 Learning the 6-multiplexer

Figure 1 shows an experiment in which ***Boole*** learned to respond correctly in the 6-multiplexer problem. The graph plots two relevant system quantities versus the number of trials since the experiment began. The upper plot shows the system's *average score*, a moving average over the past 50 trials of the percentage of the system's decisions that were correct. The lower plot shows a quantity we call the *solution count*, whose meaning requires some discussion. At an early point in our work we noticed that ***Boole*** would almost invariably tend to discover and multiply the members of a particular set of classifiers. For the 6-multiplexer, they were the following:

```
0 0 0 # # #   /   0
0 0 1 # # #   /   1
0 1 # 0 # #   /   0
0 1 # 1 # #   /   1
1 0 # # 0 #   /   0
1 0 # # 1 #   /   1
1 1 # # # 0   /   0
1 1 # # # 1   /   1
```

We named this set [S6]. Fortunately, the members of [S6] are not just any old classifiers. Inspection shows that each matches exactly eight of the 64 possible input strings and recommends the right answer each time. Furthermore, the eight matched sets are disjoint. The members of [S6] mirror the function F_6 in the sense that the four with action "1" have taxa whose specified positions match the four conjuncts of F_6, and the other four (necessary because the system must generate a "0" output overtly, not by default) match the conjuncts of the complement of F_6. There does not seem to exist a more efficient, complete set of "solution" classifiers than [S6] (and its analogues for larger multiplexers). For this reason we termed [S6] the *solution set* and used a count of its total number of instances in [P], the *solution count*, as a measure of the system's progress in evolving correct, maximal generalizations.

Returning to Figure 1, we see that by about 15,000 trials (on average about 230 exposures to each of the 64 possible input strings), the system has stabilized and shows no further progress. To summarize [P] at this point it is useful to compute the population's *macrostate*, a term borrowed from statistical thermodynamics and used here analogously. We first defined a "concept," technically, as a set of structurally identical classifiers in [P] such that all other classifiers in [P] are different from those in the set. (To avoid confusion, we shall use quotes when entering a discussion in which this technical sense is intended.) Then, to form the macrostate, we examined [P], identified each of its "concepts," and listed them in descending order of number of instances.

The result for the experiment of Figure 1 (at 15,000 trials) appears in Table 1. The top eight concepts, with the lion's share of the strength, are the members of [S6]. Many of the remaining classifiers are wrong at least part of the time. They appear to be a kind of residual noise resulting primarily from crosses between members of [S6]. Since [S6] now dominates, crossover cannot produce further improvement, yet it continues to act due to the discovery component. The implication is that the number of non-[S6] classifiers at equilibrium should be a function of χ, the crossover probability, as well as the rate at which such classifiers can be properly evaluated and deleted by the system. Furthermore, since errors can only be made by non-[S6] classifiers, the equilibrium error rate should increase with χ.

These conclusions were supported by further runs in which Figure 1's parameter regime was maintained but χ was varied. In each case, the system was operated out to 64,000 trials to ensure that it had stabilized. Then from the data were computed: (1) $\bar{a}$, the mean of the average score in the final 8,000 trials; (2) $\bar{s}$, the mean of the solution count in the final 8,000 trials; and (3) *shoulder*, defined as the number of trials at which the solution count first reaches 90% of $\bar{s}$. Table 2 shows the results (curves were qualitatively similar to Figure 1). *Shoulder* comes earlier with increasing crossover probability, suggesting that higher crossover rate -- or faster introduction of variation relative to the reproduction rate - produces faster progress. The price for this, though, is shown by the other two measures. Both $\bar{a}$ and $\bar{s}$ worsen as χ increases. This was a general result, seen in many experiments: learning speed traded off with learning reliability - at least when the rate of variation was fixed throughout an experiment.

Table 1. Macrostate of the population of Figure 1 at 15,000 trials.

Number of				*"Concept"*					*Total*
instances		*(taxon)*						*(action)*	*strength*
56	0	1	#	0	#	#	/	0	7655
52	0	1	#	1	#	#	/	1	7541
48	0	0	0	#	#	#	/	0	7056
46	1	0	#	#	0	#	/	0	7095
45	0	0	1	#	#	#	/	1	6665
41	1	1	#	#	#	0	/	0	5964
39	1	1	#	#	#	1	/	1	6323
35	1	0	#	#	1	#	/	1	5145
7	#	1	#	1	#	1	/	1	1044
4	1	1	#	#	#	#	/	1	522
3	#	0	#	#	1	#	/	1	293
3	1	1	#	#	0	#	/	0	210
2	#	0	1	#	#	#	/	1	330
2	0	1	1	#	#	#	/	1	212
2	1	0	0	#	0	#	/	0	150
2	0	0	#	#	#	#	/	1	219
2	1	#	#	#	1	#	/	1	326
2	1	0	#	0	#	#	/	0	238
1	#	1	#	0	#	#	/	0	129
1	1	0	#	#	#	#	/	0	168
1	1	1	#	#	#	#	/	0	97
1	1	0	#	#	0	0	/	0	100
1	0	0	#	#	1	#	/	1	81
1	1	#	#	#	#	0	/	0	212
1	1	0	#	#	0	1	/	1	56
1	0	1	#	#	#	#	/	1	116

The results of Table 2 may be compared with Barto's results on the 6-multiplexer task. Barto's network consisted of four A_{R-P} elements, each with inputs from each bit of the input string, and a fifth (output) unit getting its inputs from the other four plus the bits of the input string. As in our case, the input strings were generated randomly. His "reward-penalty" regime was analogous to the one we used for the Table 2 experiments. Barto states that the network achieved a 99% (in 1000 consecutive trials) performance level within 133,149 trials, on the average. The second experiment in Table 2 has $\bar{a}$ equal to 99%; in this case, *shoulder* occurred at 16,000 trials. In all four experiments of Table 2, the average score flattened out to its final range by the point at which *shoulder* occurred (as seen, e.g., in Figure 1). Thus, though the data are presented differently, it is reasonable to conclude that ***Boole*** will reach a given performance level in significantly fewer trials than the A_{R-P} network, at least on this problem.

Table 2. Effects of crossover rate χ (other parameters same as for Figure 1).

χ	*Shoulder*	$\bar{s}$	$\bar{a}$
0.03	25,500	391	99.4
0.06	16,000	388	99.0
0.12	12,000	378	98.4
0.25	9,000	353	96.7

4.3 Selection pressure

The discovery component's genetic algorithm introduces variation and reproduces high-strength classifiers, but it is the reinforcement component that determines which kinds of classifiers, or "concepts," will be selected and come to dominate [P]. If the system is to reach a state in which, for example, [S6] dominates, it is important that the reinforcement algorithm reward classifiers that lead[6] in that direction and not some other. Two kinds of "pressure" appear to be central in steering the system: pressure toward accuracy and pressure toward generality.

Pressure toward accuracy can be achieved in several ways, but basically one makes the correct or best answer pay more than other answers: the greater the difference, the greater should be the pressure toward accuracy. But one would also like a system to be robust enough to learn the best answer even when the payoff difference between the best answer and other answers is not large. We experimented with all three payoff regimes defined in steps 3 and 4 of the reinforcement algorithm. In each case the correct answer received a payoff R equal to 1000. In the *payoff-penalty* regime, error caused the removal of a fraction p (the *penalty fraction*) from the strengths of all classifiers in [A], i.e., those matching classifiers that advocated the decision actually taken. In *payoff-only*, [A] was left unaffected. In *payoff-payoff*, [A] received a positive payoff $R' < R$. Of course, the latter two are the payoff regimes we have suggested are typical of the animat's state. But to the extent the animat can determine that a particular outcome is definitely undesirable (e.g., painful), a regime like payoff-penalty could realistically apply.

We obtained pressure toward generality by biasing the distribution of payoff to [A] using the distribution function D. The idea was to pay a classifier a bit more than an equal share if it had more #'s than the average number of #'s in classifiers of [A], and a bit less if it had fewer #'s than the average. To be precise, let the ith classifier in [A] have *generality* g_i = (number of its #'s)$/L$. Define further a quantity $d_i = 1 + G \times g_i$,

[6] Initial populations contained no instances of [S6].

Table 3. Effects of selection regime (other parameters same as for Figure 1, except for $t = 0.05$ in fourth experiment).

Selection regime	*Shoulder*	$\bar{s}$	$\bar{a}$
$G = 4$ $p = 0.8$	12,000	378	98.4
$G = 1$ $p = 0.0$	none	0	50.0
$G = 0$ $p = 0.0$	23,000	370	98.5
$G = -0.4$ $p = 0.0$ $R' = 500$	25,500	284	86.4
$G = 0$ $p = 0.0$ *Noise* $= 0.1$	33,000	364	97.8

where G is a real number. Finally, let the distribution function $D(G)$ pay the ith classifier an amount

$$R_i = \left(\frac{d_i}{\sum_i d_i}\right) \cdot R \tag{2}$$

Here G controls the bias in $D(G)$. Examination will show that if $G = 0$, the distribution is uniform, while $G > 0$ favors generality.

In combination, these two pressures tended to emphasize classifiers that match in as many situations as possible while at the same time being 100% accurate (advocate the preferred answer) in those situations. If a competing classifier is equally general but less accurate, it will be weaker due to its losses where it is wrong. If a competing classifier is equally accurate but less general, it will be weaker due to receiving less from $D(G)$. There are limits, however. If $D(G)$ favors generality too strongly, then overly general classifiers, whose action is right in some situations and wrong in others, will win out. In effect, it becomes more remunerative for a classifier to add a # than to stay accurate. On the other hand, if $D(G)$ is too weak the system may not discover the most efficient "concepts," or do so only very slowly. $D(G)$ also has the property that it not only weakens classifiers that are biased against, it *drives them out.* For example, if two accurate concepts share payoff in a situation but one is more general (and $G > 0$), the less general tends toward zero members. We examine the nature of $D(G)$ further in the Appendix.

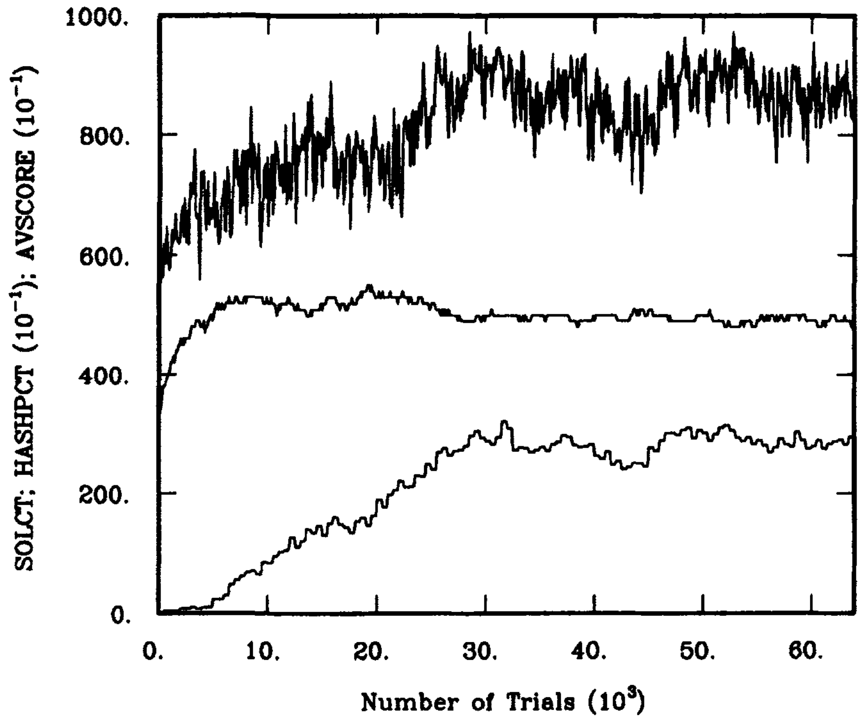

Figure 2. Experiment with the 6-multiplexer under payoff-payoff. Curves have same meaning as in Figure 1. Values of system parameters are the same as in Figure 1, except $G = -0.4$, $R' = 500$, $p = 0$, $t = 0.05$.

Effects of selection regime are illustrated by five experiments shown in Table 3. In the first experiment, a strong generalization "pull" was played off against a substantial penalty for error. As indicated by *shoulder*, the solution count climbed relatively rapidly to a high asymptotic value. The second experiment, an example of payoff-only, shows what happened when the penalty was eliminated: even at lower G, generality won out over accuracy and [P] became dominated by overly general classifiers. In the third experiment both G and p, the penalty fraction, were zero; the solution set [S6] took over [P], but more slowly and less completely than under payoff-penalty. Since $G = 0$, this experiment suggests that a "natural" generalization bias of some strength exists in the system. (The Appendix discusses a possible source of this bias.)

The fourth experiment exemplifies payoff-payoff. Besides getting payoff of 1000 for right answers, ***Boole*** got payoff 500 for wrong answers. To succeed, it had in effect to determine statistically that right answers paid more, and to emphasize the corresponding classifiers. This happened, but not without some struggle, as is illustrated by the curves of Figure 2. Note that $G = -0.4$ in this experiment. Negative G's, which cause $D(G)$ to favor specific classifiers over general ones, were necessary to get progress under payoff-payoff. With $G = 0$, the system became overgeneral and the average score stayed at chance levels. Because errors receive positive reinforcement under payoff-payoff, it took little surplus generalization pressure to give overly general classifiers the edge. Apparently, the progress in the fourth experiment was due to *offsetting* part of the natural generalization pressure with negative G.

The last experiment in Table 3 had the same parameters as the third experiment except that payoff was made "noisy." With probability 0.1, the environment withheld payoff when the system's answer was correct and gave payoff when the answer was wrong. Though learning took longer, the average score and solution count reached nearly the same final levels as in the third experiment, indicating that ***Boole*** can overcome this amount of classification noise.

Step 5 of the reinforcement algorithm deducts a fraction t from the strengths of NOT[A], the members of the match set that *did not* advocate the decision made by the system. This is a different sort of selection pressure than that exerted by the payoff regime of steps 3 and 4. Experimentally, the effect was to push the system to decide more quickly which answer it preferred in each matching situation. Reducing the strength of NOT[A] increases the relative strength of classifiers in [A], and their chance of constituting [A] the next time. The resulting instability tended eventually (other things being equal) to cause one set to disappear. As long as t was small, say 0.1, the survivor was usually the correct one, and the overall rate of evolution was increased. Usually, the average score also improved.

To summarize this section: ***Boole*** learned most quickly and decisively under payoff-penalty and least so under payoff-payoff. Explicit generalization pressure can be exerted, but there is an inherent pressure as well. In the presence of considerable classification noise, the system will still converge on the correct classifiers. Within limits, performance can be improved by deliberately weakening classifiers that disagree with the system's decision.

4.4 Crossover vs. mutation

We now turn to a set of experiments that sought to determine the relative contributions of crossover and mutation to the discovery component. Mutation is a conservative operator, since a mutated classifier is a small step in classifier space from its parent. In contrast, a crossover offspring can be (though need not be) a large jump from its parents. Similarly, in the space of classifier substrings, mutation introduces a set of small steps away from a classifier's constituent substrings, whereas one cross both creates numbers of novel substrings and tests extant substrings in many new contexts. From considerations of this sort, Holland (1975) suggested that crossover is the primary discovery operator in natural systems and that mutation serves as a background operator needed only to preclude the possible permanent loss of alleles. De Jong (1975) confirmed the superiority of crossover in experiments using the genetic algorithm for function optimization. We used the 11-multiplexer problem to investigate the question in classifier systems.

The 11-multiplexer is a good setting from comparing crossover and mutation because the "solution set" [S11] is a far smaller subset of its classifier space than is [S6] of its space. Since the search problem is more difficult, any advantage of crossover over mutation should be easier to bring into relief. The space of the 6-problem contains $2 \times 3^6 = 1458$ classifiers of which 8, or 0.55% belong to [S6]. The space of the 11-problem contains $2 \times 3^{11} = 354,294$ classifiers of which 16, or 0.0045% belong to [S11]. Members of [S11] are analogous to those of [S6]; an example is

$$0\ 1\ 0\ \#\ \#\ 1\ \#\ \#\ \#\ \#\ \#\ \ /\ \ 1\ .$$

In comparing crossover and mutation, there would seem to be two possibilities: (1) crossover adds search power that cannot be gained by any amount of mutation; or (2) crossover is just another way of "stirring the pot" and is not more powerful than some equivalent amount of mutation. To investigate this we had ***Boole*** learn the 11-multiplexer under various combinations of crossover and mutation probabilities, as shown in Figure 3. Other parameters were the same as in Figure 1, except that the genetic algorithm was invoked with probability $\rho = 0.25$ per trial. Figure 3 shows the solution count reached in each case after 64,000 trials.

The largest entry in Figure 3 is a solution count of 297 for $(\chi, \mu) = (0.25, 0.0025)$. That value is significantly larger than for any experiment with crossover equal to zero, even though μ was varied over a wide range. The implication is that crossover indeeds adds search power that cannot be provided by any amount of mutation. A further implication, from the points with $\chi = 0.25$, is that the optimal value for the mutation probability is around 0.001 or 0.0025, close to the reciprocal of population size (and so possibly related to a similar result found by De Jong for populations of size 50).

4.5 Variation control

In Section 4.2 we noted the "noise" classifiers that remained even after a population had converged on the solution set and stabilized. The noise

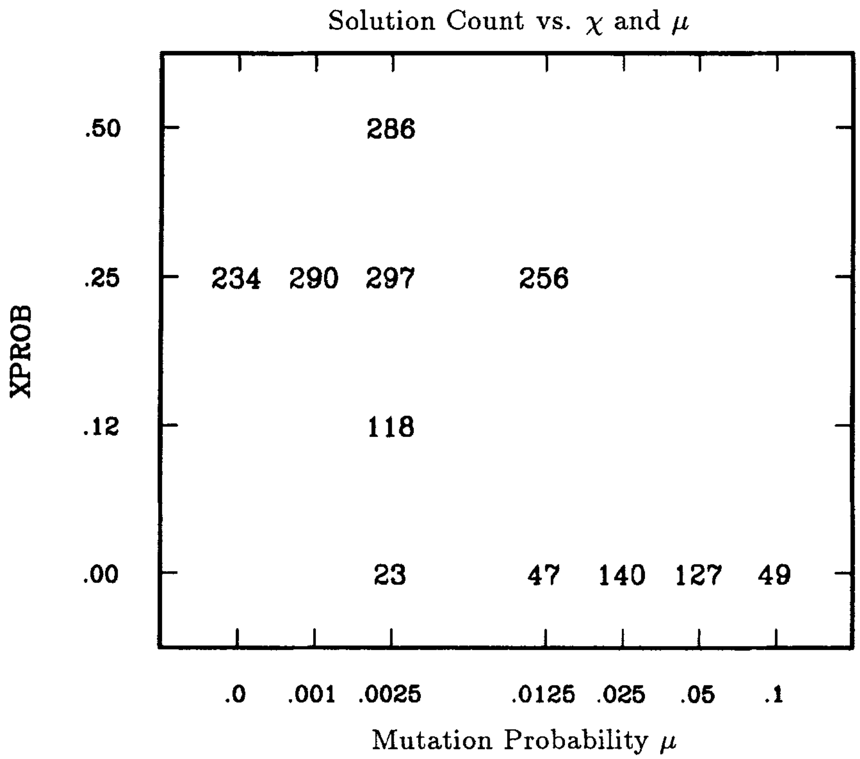

Figure 3. Solution count for the 11-multiplexer at 64,000 trials for various values of crossover and mutation probabilities. Values of other system parameters are the same as in Figure 1, except $\rho = 0.25$.

was due to the continued, but no longer necessary, introduction of variation by the discovery component. Yet "noise" (crossover and mutation) *was* necessary earlier, in order to find the solutions in the first place. If the system had access to the solution count - which of course it does not - it could in principle itself control the rate of variation. This would let it progress rapidly in the beginning but reduce exploration later on, avoiding the trade-off between learning speed and reliability. In this section we suggest that the system may be able to achieve this objective by using an environment-independent measure of its own entropy.[7]

To control variation profitably, the system needs both a control policy and a measure of system progress. There are many conceivable control

[7] See Booker (1985) for a different approach to variation control, in which the relative probability of crossing depends on the similarity of the potential parents. Also see Baker (1985) on premature convergence.

policies but there appear to be few fundamental, computable, environment-independent measures of progress. For the latter, we chose to experiment with entropy because it measures "uncertainty" and "disorganization," both of which a classifier system presumably reduces as it learns. In addition, classifiers exist in a space formally similar to the "phase space" of statistical mechanics (Sears, 1953) so the application of the entropy concept is straightforward and plausible.

The space of ***Boole***'s classifiers has $L + 1$ dimensions: L for each taxon position and one more for the action. The possible values on a taxon dimension are 0, 1, and #, while on the action dimension the values are just 0 and 1. Each point in this space defines a possible classifier. The actual classifiers of a particular population [P] may be thought of as occupying a particular set of points in classifier space. If two classifiers in [P] are identical, they occupy the same point. More generally, the classifiers of a "concept" all occupy the same point. From this perspective, [P]'s macrostate (Section 4.2) is a listing of the occupancy numbers and total strengths of points occupied by [P].

The entropy of such a distribution of classifiers over points in classifier space may be expressed by the Shannon (1948) entropy

$$H_s = -K \sum_i p_i \ln p_i \qquad ,$$

if the p_i are taken equal to N_i/N, where N_i is the number of classifiers occupying the ith occupied point (concept) and N is the total number of classifiers in [P]. K is an arbitrary constant. Because concept strengths anticipate occupancy numbers and tend to be proportional to them, we prefer to take the p_i as S_i/S_T, the concept strength divided by the total strength of [P]. We then define *classifier system entropy* as

$$H_c = \frac{-\sum_i (S_i/S_T)\ln(S_i/S_T)}{\ln N} \qquad .$$

The factor $1/(\ln N)$ is a normalization in the sense that if all classifiers in [P] are different and of equal strength, then the system has maximal uncertainty and $H_c = 1$.

Experimentally, we found that H_c does indeed fall as the solution count rises. In fact, H_c begins falling before any solution classifiers appear, reflecting the development of precursor concepts. If the solution count eventually reaches a plateau, so does H_c. In an experiment in which no strong concepts develop, the entropy falls little. But H_c is not perfect: it falls rapidly if [P] becomes overgeneral; and it can assign a low entropy to a state in which only *some* solutions have been found. H_c measures order, which does not always equal progress. For example, in the state of lowest entropy all classifiers are identical.

Nevertheless, H_c can permit an effective control of variation that gives faster and more complete learning than under fixed variation rates. We experimented with control policies that increased crossover as long as entropy was falling (***Boole*** is "absorbing" variation, so give it more), but reduced crossover if entropy was flat or rising (let the system digest the current diversity). A specific algorithm (executed every 100 trials) was:

1. Measure H_c. Compare it with the previous value and compute the change;
2. If the change is less than Δ_1 (a negative number), increment χ, the crossover probability, by Γ_1 percent; else if the change is greater than or equal to Δ_2, decrement χ by Γ_2 percent.

Figure 4 shows an experiment using the 11-multiplexer. Control parameters were: $\Delta_1 = -0.010$, $\Delta_2 = 0.007$, $\Gamma_1 = \Gamma_2 = 10$; χ was initialized at 0.25. Curves of the solution count for high and low fixed χ (0.50 and 0.12) are included for comparison. The controlled solution count curve tends to have the best of both regimes: rapid growth early and low noise later on.

Entropy-based variation control is experimental at this stage; the parameters of the algorithm must be chosen carefully. But the technique appears to offer a promising kind of system self-control.[8] Variation control in classifier systems is analogous to "temperature" control in networks using simulated annealing (Kirkpatrick et al., 1983), though the latter process is based on a definition of network "energy" and is seldom made automatic.

5. Discussion

The results with our specialized classifier system are in several respects positive for the general application of classifier systems to the animat problem. With respect to learning speed, the results raise questions that should be addressed by future research.

The positive results center around the fact that ***Boole*** learned a rather complex, disjunctive concept, incrementally, under payoff regimes that included both noisy payoff and partial reward for incorrect performance ("payoff-payoff"). As discussed in the beginning, these conditions are characteristic of the animat learning problem, which occurs for animals and autonomous robots. In addition, the classifiers evolved by ***Boole*** corresponded closely to the task's expression in disjunctive normal form, making it easy

[8] This may also prove useful in applications of the genetic algorithm to function optimization (Grefenstette, 1985).

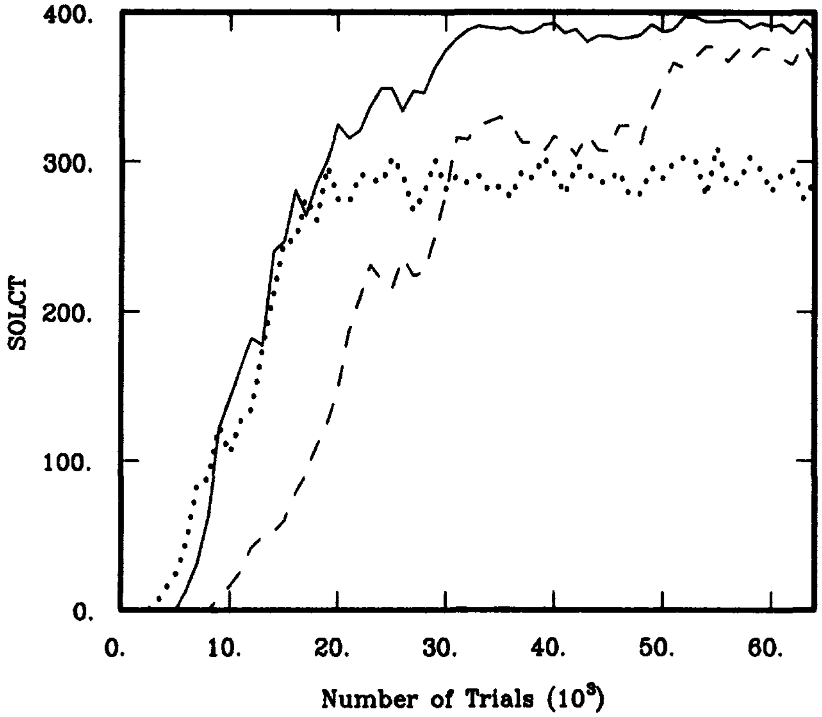

Figure 4. Solution count for the 11-multiplexer with and without automatic crossover rate control. For the dotted curve, crossover probability χ was fixed at 0.5; for the dashed curve, χ was fixed at 0.12; for the solid curve, χ was initialized at 0.25 and then controlled according to the algorithm in the text. Values of other system parameters are the same as in Figure 1.

to "see the system's knowledge" and suggesting the hypothesis that ***Boole*** will tend in general toward the disjunctive normal form. The ability to learn under payoff-payoff is important for the extension of our results to the standard classifier system, since the reinforcement regime within the bucket brigade takes this form. The experiments also illustrated, for classifier systems, the theoretically predicted superiority of crossover to mutation. In comparison with Barto's network solution to the same task, ***Boole*** learned in substantially fewer trials, possibly because it was able to recombine tentative solution hypotheses via crossover, while the network employed only a kind of point mutation. Finally, an environment-independent method by which ***Boole*** could advantageously control its own variation (crossover) rate was suggested and successfully demonstrated, illustrating a primitive form of system self-judgement.

Against these plusses was the fact that ***Boole*** progressed quite slowly in absolute terms under all payoff regimes, and was slowest under payoff-payoff. Learning times for the 11-multiplexer were somewhat longer than for the 6-multiplexer (compare Figures 1 and 4), though not dramatically so. To probe farther, we did a preliminary experiment with the 20-multiplexer, for which there are 2^{20} input strings and 2×3^{20} possible classifiers. Under payoff-penalty, the system reached an average score above 90% within 70,000 trials; by 120,000 trials, the solution count (there are 32 solution "concepts" for this problem) reached 1200 out of a population size of 1600. This seems quite remarkable given that by 120,000 trials ***Boole*** had seen, on average, less than 1/8 of the possible input strings. Thus, it appears ***Boole*** can solve the multiplexer for longer and longer strings. But the learning times clearly increase, and in this experiment we had to increase the population size to get learning started.

The implication of these problems for the standard classifier system is that as classifiers become more complex, slowness may compound and populations may get very large. Bucket-brigade chains may grow slowly due to slow learning in each link. Furthermore, the less definite learning under payoff-payoff may make long, reliable chains hard to set up.

These are important points. But both computer science and nature suggest a solution via modularity. In structured programming languages, control sequences (and indeed whole programs) can be expressed as a short sequence of steps wherein each step is itself expandable into a short sequence of substeps, etc. Thus potentially extensive and complex behaviors (e.g., "going to the opera" or "finding and eating a zebra") can be described, down to the motor signal level, in terms of a hierarchy of modules each consisting of a few steps. Albus (1979) presents a behavioral model having this form, and there is clearly a relation to problem-solving systems employing subgoals (Laird et al., 1986). Our suggestion is that research on classifier systems aim at ways of inducing bucket-brigade chains whose message sequences "name" subchains, which in turn name or call subchains, etc. Holland (1985) considers this question, and we have offered a proposal (Wilson, in press). In addition, modularity should permit simpler classifiers, since the essence of a module is that it is nearly context-independent and so has a simple activation condition.

Keeping both classifiers and bucket-brigade chains short through modularity or hierarchy offers promise of overcoming the slowness that our results suggest may occur in extensive applications of classifier systems. The details of implementing a classifier system with hierarchical structure are a problem for future research. The present work's principal contribution has been to show that a classifier system can learn a difficult logical task under realistically difficult environmental constraints.

Acknowledgements

I would like to thank Pat Langley and Ryszard Michalski for their helpful comments on earlier drafts of the paper.

References

Albus, J. S. (1979). Mechanisms of planning and problem solving in the brain. *Mathematical Biosciences*, *45*, 247–293.

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R., & Kline, P. J. (1979). A learning system and its psychological implications. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp. 16–21). Tokyo, Japan: Morgan Kaufmann.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (pp. 101–111). Pittsburgh, PA.

Barto, A. G. (1985). *Learning by statistical cooperation of self-interested neuron-like computing elements* (COINS Technical Report 85–11). Amherst: University of Massachusetts, Department of Computer and Information Science.

Booker, L. (1982). *Improving behavior as an adaptation to the task environment.* Doctoral dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

Booker, L. (1985). Improving the performance of genetic algorithms in classifier systems. *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (pp. 80–92). Pittsburgh, PA.

Curtis, H. (1983). *Biology.* New York: Worth.

Davis, R., & King, J. (1977). An overview of production systems. In E. W. Elcock & D. Michie (Eds.), *Machine intelligence* (Vol. 8). New York: American Elsevier.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

Goldberg, D. E. (1983). *Computer-aided pipeline operation using genetic algorithms.* Doctoral dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

Grefenstette, J. J. (1985) (Ed.). *Proceedings of an International Conference on Genetic Algorithms and Their Applications.* Pittsburgh, PA.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Holland, J. H. (1976). Adaptation. In R. Rosen & F. M. Snell (Eds.), *Progress in theoretical biology* (Vol. 4). New York: Plenum.

Holland, J. H. (1985). Properties of the bucket brigade algorithm. *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (pp. 1–7). Pittsburgh, PA.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Holland, J. H., & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In D. A. Waterman & F. Hayes-Roth (Eds.), *Pattern-directed inference systems*. New York: Academic Press.

Kirkpatrick, S., Gelatt, C. D., Jr., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671-680.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11-46.

Michalski, R. S. (1986). Understanding the nature of learning: Issues and research directions. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Michalski, R. S., & Larson, J. B. (1978). *Selection of most representative training examples and incremental generation of VL_1 hypotheses: The underlying methodology and the description of programs ESEL and AQ11* (Technical Report No. 867). Urbana: University of Illinois, Computer Science Department.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, *18*, 203-226.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (ICS Report 8506). San Diego: University of California, Institute for Cognitive Science.

Schaffer, J. D. (1985). Learning multiclass pattern discrimination. *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (pp. 74-79). Pittsburgh, PA.

Sears, F. W. (1953). *Thermodynamics*. Reading, MA: Addison-Wesley.

Shannon, C. E. (1948). The mathematical theory of communication. *Bell System Technical Journal*, *27*, 379-423, 623-656.

Smith, S. (1980). *A learning system based on genetic algorithms*. Doctoral dissertation, Department of Computer Science, University of Pittsburgh, PA.

Waterman, D. A. (1970). Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence*, *1*, 121-170.

Wilson, S. W. (1985). Knowledge growth in an artificial animal. *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (pp. 16-23). Pittsburgh, PA.

Wilson, S. W. (in press). Hierarchical credit allocation in a classifier system. In L. D. Davis (Ed.), *Genetic algorithms and simulated annealing*. London: Pitman.

Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill.

Appendix

We discuss here some heuristic approaches to understanding the generalization and "drive out" effects that occur in ***Boole***.

The macrostate of [P] (see Section 4.2) is a partition of the population into "concepts" $[\mathrm{C}]_i$, each consisting of N_i identical classifiers with total strength S_i. The N classifiers of [P] have total strength S_T. There is a useful rule of thumb, supported empirically, which says that any two concepts $[\mathrm{C}]_1$ and $[\mathrm{C}]_2$ tend toward an equilibrium in which

$$S_1/S_2 = N_1/N_2 \quad . \tag{A1}$$

The rule may be justified by noting that, at equilibrium, the probability P_R that $[\mathrm{C}]_i$ will reproduce (add one member) at the next invocation of the genetic algorithm must equal the probability P_D that one member will be deleted. (For simplicity we ignore crossover and mutation.) Thus we have

$$P_R = S_i/S_T \quad . \tag{A2}$$

Now, if deletion is a uniform random process, then

$$P_D = N_i/N \quad , \tag{A3}$$

so that at equilibrium,

$$S_i/S_T = N_i/N \quad ,$$

which implies (A1). However, if the probability of deleting a classifier is proportional to the reciprocal of its strength (as in this study),

$$P_D = \frac{N_i^2/S_i}{\sum_k(N_k^2/S_k)} \quad , \tag{A4}$$

where we assume all classifiers in a concept have equal strength. Then, at equilibrium, (A2) and (A4) imply

$$(S_i/N_i)^2 = \frac{S_T}{\Sigma_k(N_k^2/S_k)} \quad , \tag{A5}$$

which also implies (A1), since the right side is independent of i. Derivation of (A1) requires the assumption that an equilibrium is possible. This is reasonable for classifier systems, since the payoff to a concept is limited and thus the strength is as well. (An analogous assumption definitely does not hold in the case of systems using the genetic algorithm for function optimization. This is an important difference between the two uses of the algorithm.)

Equation (A1) states two slightly different things. First, it says the size (number of members) of a concept is proportional to its strength. But it also says that individual classifiers in different concepts tend to have the same strength, which is indeed a frequent observation in populations that are not changing rapidly.

A further implication of equation (A1) is that, in the absence of other forces, any concept whose strength is greater than zero will persist in [P]. This is desirable in that two concepts which do not compete for the same payoffs - for example, two members of [S6] will not place pressure on each other, thus permitting [P] to develop solutions to basically separate subproblems. However, it is undesirable in that two concepts which *do* compete for the same payoffs - for instance, a member of [S6] and a more specific version of the same will exist in some sort of equilibrium, whereas one might prefer a situation in which the more general one "drove the other out."

The distribution function D overcomes this drawback, permitting more general classifiers to win out over less general ones, as long as their accuracies are equal. (The effect is not restricted to generality; D can be designed to favor any other characteristic that depends on a classifier's formal structure.) To see this, assume $[\mathrm{C}]_1$ and $[\mathrm{C}]_2$ share payoff in a certain situation and that they do not get payoff anywhere else. Further assume that D gives payoff shares P_1 and P_2, respectively, to members of each concept, in accordance with equation (2) (Section 4.3). Using the basic reinforcement equation (1) (Section 3.3), we can then write:

$$S'_1 = S_1 - eS_1 + \left(\frac{N_1 d_1}{N_1 d_1 + N_2 d_2}\right) P \quad \text{and} \quad S'_2 = S_2 - eS_2 + \left(\frac{N_2 d_2}{N_1 d_1 + N_2 d_2}\right) P \quad ,$$

where the (unequal) d's contain the generalization biases. These equations imply that $[\mathrm{C}]_1$ and $[\mathrm{C}]_2$ will tend toward equilibrium strengths

$$S_1 = \frac{N_1 d_1}{N_1 d_1 + N_2 d_2}\left(\frac{P}{e}\right) \quad \text{and} \quad S_2 = \frac{N_2 d_2}{N_1 d_1 + N_2 d_2}\left(\frac{P}{e}\right) \quad .$$

The ratio of strengths is:

$$S_1/S_2 = (N_1/N_2)(d_1/d_2) \quad . \tag{A6}$$

However, this ratio is not consistent with equation (A1), required by the reproductive process, except if $S_1 = N_1 = 0$ or $S_2 = N_2 = 0$; that is, by elimination of one or the other concept. If $[\mathrm{C}]_1$ is more general than $[\mathrm{C}]_2$, the bias will favor it and $[\mathrm{C}]_2$ will be driven out.

Now let us assume that $[\mathrm{C}]_1$ is overgeneral; that is, other situations exist in which $[\mathrm{C}]_1$ is incorrect (or, under payoff-payoff, has lower payoff). What is the effect? We can get a crude idea by letting fS_1, with f a factor, represent the relative loss suffered by $[\mathrm{C}]_1$ in these other situations. Following the same steps as in deriving (A6), we get

$$S_1/S_2 = (N_1/N_2)(d_1/d_2)(e/(e+f)) \quad . \tag{A7}$$

In this case, the instability will favor $[\mathrm{C}]_1$ or $[\mathrm{C}]_2$, depending on whether

$$(d_1/d_2)(e/(e+f))$$

is greater or less than unity.

Apart from effects of the distribution function D, certain equilibria implied by (A1) are under pressure from the deletion operator, which affects less active concepts relatively more than more active ones. Deletion of a classifier c from $[\mathrm{C}]_1$ reduces $[\mathrm{C}]_1$'s strength by the strength of c. If $[\mathrm{C}]_1$ is active, i.e., receives frequent payoffs relative to the reproduction rate, its total concept strength – and therefore, by reproduction, the lost classifier – will be quickly made up. But a less active concept will take longer to restore the strength and lost member. Now assume $[\mathrm{C}]_1$ and $[\mathrm{C}]_2$ are often in [A] together – that is, they frequently compete for the same payoffs – but that $[\mathrm{C}]_2$ is less general and therefore less active than $[\mathrm{C}]_1$. For the reason just given, whatever the ratio N_1/N_2 may be at some common payoff event, by the next one it is likely to be larger. However, this will "steer" a larger payoff share to $[\mathrm{C}]_1$, causing a further increase in N_1/N_2 through reproduction. The cycle repeats until $[\mathrm{C}]_2$ is eliminated. We hypothesize that this process is responsible for the "natural" generalization pressure noted in Section 4.3.

The deletion effect probably also helps the system "decide" which answer to give in a matching situation, i.e., which concept in [M] should win out. The more remunerative concept has the higher probability of constituting [A], but this forces relative inactivity on the other members of [M]. The deletion effect then makes the latter even less likely to get in [A], which leads finally to their being "driven out." Note that both these cases depend on a specific mechanism by which any ascendancy of one concept takes payoff away from the other.