Machine Learning 3: 139-159, 1988
© 1988 Kluwer Academic Publishers - Manufactured in The Netherlands

A Tale of Two Classifier Systems

GEORGE G. ROBERTSON[†] (ROBERTSON.PA@XEROX.COM)
Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142, U.S.A.

RICK L. RIOLO (RICK_RIOLO@UM.CC.UMICH.EDU)
Electrical Engineering & Computer Science Department, University of Michigan, Ann Arbor, MI 48109, U.S.A.

(Received: December 1, 1987)
(Revised: May 12, 1988)

Keywords: Classifier systems, genetic algorithms, parallelism, sequence prediction

Abstract. This paper describes two classifier systems that learn. These are rule-based systems that use genetic algorithms, which are based on an analogy with natural selection and genetics, as their principal learning mechanism, and an economic model as their principal mechanism for apportioning credit. CFS-C is a domain-independent learning system that has been widely tested on serial computers. *CFS is a parallel implementation of CFS-C that makes full use of the inherent parallelism of classifier systems and genetic algorithms, and that allows the exploration of large-scale tasks that were formerly impractical. As with other approaches to learning, classifier systems in their current form work well for moderately-sized tasks but break down for larger tasks. In order to shed light on this issue, we present several empirical studies of known issues in classifier systems, including the effects of population size, the actual contribution of genetic algorithms, the use of rule chaining in solving higher-order tasks, and issues of task representation and dynamic population convergence. We conclude with a discussion of some major unresolved issues in learning classifier systems and some possible approaches to making them more effective on complex tasks.

1. Introduction

Learning classifier systems acquire rules, called *classifiers*, to perform some specified task (Holland, 1986; Holland & Burks, 1987; Holland & Reitman, 1978). The task is specified by a set of examples presented to the system and an evaluation function that determines how well the system is performing the task. The evaluation function provides sparse reinforcement (a scalar reward or punishment) rather than providing correct answers. Rules are message-oriented, taking messages as inputs and producing as outputs new messages that perform the task and control the sequence of rule activation. Each rule

[†]Author's current address: Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, U.S.A.

has an associated strength that indicates its effectiveness in performing the desired task. In classifier systems, apportionment of credit is done by the *bucket brigade*, an algorithm for adjusting rule strengths based on an analogy with a service economy (Holland, 1985a). New rules are created by a *genetic algorithm* (Holland, 1975), which is based on an analogy with genetics and natural selection. Most components of such systems, including rule and message management, strength adjustment, and the learning mechanisms, are inherently parallel.

This paper focuses on experimental results obtained with two particular learning classifier systems, CFS-C and *CFS. The first part of the paper describes the differences between these systems and the standard model of learning classifier systems. The second part illustrates the current state of classifier systems and genetic algorithms by discussing several empirical studies of the behavior of these two systems. These studies examine some known problems with the classifier system paradigm. The first study examines the issue of population size. The second explores the actual contribution of genetic algorithms to the system's learning behavior. The third study focuses on chains of rules, which are necessary to solve higher-order tasks. The final study deals with representational issues and the convergence of the population over time.

In general, like other approaches to machine learning, existing classifier systems behave well on simple and moderately-sized tasks, but they break down on larger, more complex tasks. However, no fundamental barriers have yet been encountered while moving toward these larger tasks. The studies and discussion in this paper illustrate the current limits of classifier systems, but they also suggest modifications that may make them more effective on complex tasks.

2. Two learning classifier systems

All of the experiments described in this paper were carried out on two classifier systems, CFS-C and *CFS. CFS-C is implemented in the C programming language and runs on a number of serial computers (Riolo, 1986a). *CFS is a parallel implementation of CFS-C on the Connection Machine (Hillis, 1985), a massively parallel computer with 65,536 processors, with one classifier assigned to each processor. The *CFS implementation, described in Robertson (1987), demonstrates that most aspects of learning classifier systems can be implemented in parallel. Both systems follow the standard *Michigan Approach* (developed by John Holland and his associates at University of Michigan) to classifier systems (De Jong, 1988; Holland, 1986). Although CFS-C and *CFS differ in minor ways, they both contain the same major features, which has made it possible to run many of the experiments reported in this paper on both systems. These two systems co-evolved, and were validated against each other before conducting the studies reported here (Robertson & Riolo, 1987). The rest of this section briefly describes the major features of CFS-C and *CFS, with emphasis on differences from the standard framework. For more details, see Robertson and Riolo (1987).

2.1 The interface to the environment

In order to define a task domain for CFS-C and *CFS, we supply *detectors* to encode the external environment, *effectors* to manipulate the environment, and an *evaluation function* to measure how effectively the system is performing. Detectors produce *messages* that are placed on a *message list* along with messages produced during the previous step. Effectors interpret some messages produced by classifiers as instructions about how to manipulate the environment. The results of effector actions are judged by the evaluation function, which returns a single *payoff* value as feedback for each cycle.

2.2 Classifiers and messages

Messages in CFS-C and *CFS are fixed-length binary strings. The conditions and action of each classifier are ternary strings (over {0, 1, #}) of the same length as messages. Although in general a classifier is a rule with one action and an arbitrary number of conditions, both CFS-C and *CFS simplify this to two conditions and one action. This can be done without loss of generality, provided chains of rules can be formed and maintained.

A classifier is matched when both conditions are satisfied. If the second condition is negated, it is satisfied if no message matches it. Matched classifiers compete by *bidding* to post messages to a fixed-size message list, and those that win the (probabilistic) competition produce new messages using the "pass through" operator. A classifier's bid is proportional to its *strength* times its *specificity* (more specific rules bid more).

2.3 Credit assignment algorithms

In both CFS-C and *CFS, strength is primarily allocated by the *bucket brigade algorithm (BBA)*. The basis for the BBA is feedback from the environment: *all* classifiers that post messages during a given step have the payoff from the environment added to their strength. The BBA also redistributes strength from classifiers to other classifiers, as each classifier that posts messages pays the amount it bid to the classifiers that made it possible for it to become active. Riolo (1987a, 1987b) discusses some of the issues involved in making the BBA effective.

In addition to the BBA, CFS-C and *CFS also apply *head*, *bid*, and *producer taxes* to adjust classifier strengths. The head tax is generally a low fixed-rate tax applied to every classifier on every step. The head tax is necessary to reduce the strength of classifiers that are never activated (and therefore never affected by the BBA) so that they will eventually be replaced. The bid tax is a low fixed-rate tax applied to every classifier that bids, and the producer tax is a progressive tax that increases with the number of messages posted by a classifier during a given step. The main purpose of the bid and producer taxes is to control overgeneralization, by taking strength away from general classifiers that tend to bid too often or produce too many messages.

2.4 Rule discovery heuristics

CFS-C and *CFS employ three heuristic procedures to create new rules: (1) a *genetic algorithm*; (2) *cover detector* and *cover effector* operators, which force the system to be more responsive to its environment; and (3) a *triggered chaining operator*, which introduces fragments of rule chains into the system.

Genetic algorithm. CFS-C and *CFS employ a standard genetic algorithm (GA) using the crossover and mutation operators. The fitness measure is strength: when the GA is applied (usually once every 10–20 steps), a small fraction of the high-strength classifiers are chosen to produce offspring and an equal number of low-strength classifiers are replaced. Crossover is applied to some of the new offspring by treating the entire classifier as a chromosome and carrying out either a single or a double crossover. The mutation operator is then applied to a small fraction of the new classifiers.

Cover detector and cover effector operators. When a classifier system is unresponsive to the environment (i.e., its classifiers fail to match *any* detector messages or to activate *any* effectors), the BBA will have nothing to work with and the classifier strengths will convey no useful information to the GA. To overcome this problem, CFS-C and *CFS employ the two additional operators. The cover detector operator (CDO) (Holland & Reitman, 1978; Wilson, 1985), triggered when a detector message is not matched by any classifier, responds by creating a classifier that matches the detector message and has a random action. The cover effector operator (CEO), triggered when no effector is activated, copies a classifier that is responsive to the current situation and gives it a new, random action.

Triggered chaining operator. A classifier C_1 is said to be *coupled* to classifier C_2 when a message produced by C_1 satisfies some condition of C_2. Sequences of coupled classifiers, sometimes called *classifier chains*, are necessary so that classifiers can be used to implement arbitrary networks and to perform arbitrary computations (Holland, 1986). For example, chains are necessary to implement a short-term memory, so that classifier systems can perform actions that are not entirely determined directly by messages from the environment (Booker, Goldberg, & Holland, in press).

Although the BBA has been demonstrated to properly allocate strength to classifier chains (Riolo, 1987b), in practice such chains are very rarely created by the GA. Therefore the *triggered chaining operator* (TCO), suggested by Holland, Holyoak, Nisbett, and Thagard (1986), was added to CFS-C and *CFS. The basic purpose of this operator is to create a pair of *coupled* rules that reflect an initially *accidental* activation of one rule after another. Like all classifiers, the coupled pairs produced by the TCO are evaluated and assigned credit (or blame) by the BBA. If the coupled rules improve performance (and so gain strength), they serve as building blocks for longer chains and other more complicated structures. If the coupled rules do not improve performance, they lose strength and are replaced.

In order to bias the production of coupled classifiers toward plausibly useful chains, the TCO is activated only when two conditions are met: (a) a classifier

C_2 makes a *profit*, and (b) there was a classifier C_1 active just prior to classifier C_2 that is not already coupled to C_2. We define the *profit*, $P_i(t-1)$, of classifier i at step $t-1$ as the reward it received at $t-1$ less the bid it paid at $t-1$, *plus* the amount paid to it by other classifiers active at step t.

For example, consider the following two classifiers, A and B, active at steps $t-2$ and $t-1$, respectively:

$$\begin{aligned}(t-2): &\quad A = aaaa, aaaa/xxxx\\ (t-1): &\quad B = bbbb, bbbb/yyyy.\end{aligned}$$

If $P_B(t-1) > 0$ and if A and B are not already coupled, then the TCO would produce the following new classifiers:

$$\begin{aligned}A' &= aaaa, aaaa/mmmm\\ B' &= bbbb, mmmm/yyyy,\end{aligned}$$

where *mmmm* is a random string with no predefined meaning. The result is a pair of coupled classifiers constructed so that if the conditions that caused A to fire are again encountered, A' also fires and produces a message *mmmm*, which may cause classifier B' to fire on the next step, which in turn produces a message *yyyy* that was associated with making a profit. If *yyyy* does indeed consistently lead to profit, the BBA will allocate strength to the coupled rules and they will survive and act as building blocks for longer chains.

3. Empirical studies of classifier systems

Genetic algorithms have been studied for the last twenty-five years, and they have been examined in the context of classifier systems for the last ten years. Although some substantial theory has been developed for pure GA (Goldberg, 1985; Holland, 1975), less has been developed for classifier systems (Holland, 1985a, 1985b; Wilson, 1987). The empirical studies we report here explore some areas that currently lack a strong theoretical foundation: the effects of population size, the effectiveness of GA in systems that do not promote speciation, the mechanisms necessary for effective rule chaining, and issues of convergence and representation.

3.1 The task of letter sequence prediction

The primary task domain we have used for tuning and analyzing both CFS-C and *CFS is letter sequence prediction (Riolo, 1986b). A sequence of letters of any length is chosen and presented repetitively to the system. The sequence is viewed by the system through a fixed-sized window, which is four letters for most of the studies reported here. That is, as the system moves through the sequence of letters, it views the current letter and the three previous letters, from which it must guess the next letter in the sequence. This task is represented with nine-bit messages, which use five bits to represent a letter and four bits to indicate the type of the message (i.e., one of four detectors, a prediction, or a rule chain message).

Although this task domain may at first appear to be trivial, note that the system has no *a priori* knowledge of letters, their relationships, or the notion

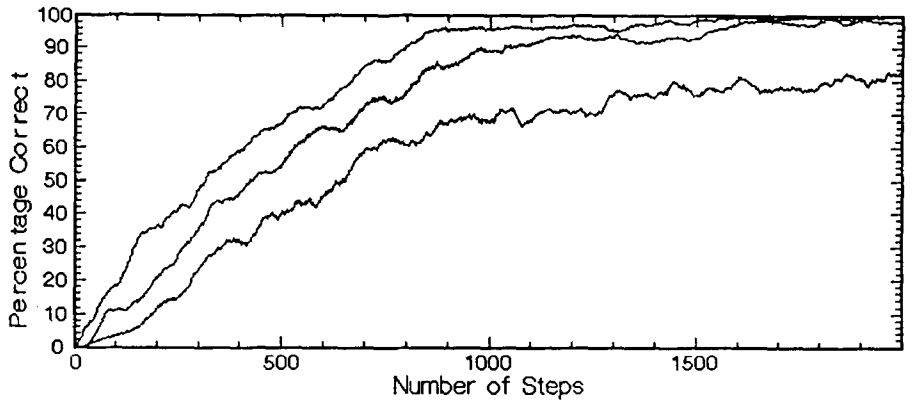

Figure 1. Learning the letter sequence "abcd" over 2000 steps. Population sizes are 50 (lowest curve), 500 (middle), and 8000 (upper).

of sequence. To illustrate the difficulty of this task, consider the size of the solution space (i.e., the number of possible classifiers) that must be searched. Since each locus can contain one of three alleles and the second condition may be negated, the number of possible classifiers for l loci is $N = 3^l \times 2$. For these problems, $l = 27$, hence N is approximately 1.5×10^{13}. Of course, many of these classifiers are phenotypically alike (i.e., they respond to the same situation in the same way), but the search space is still quite large.

3.2 Experimental procedures

Both CFS-C and *CFS are stochastic and use pseudo-random number generators in numerous places, which results in significant variance from one individual run to another. De Jong (1975) reports similar observations, and suggests averaging the results of a minimum of five runs for each experiment. We average ten runs in the results reported here. The figures shown in the following sections are comparisons of smoothed learning curves. Each point represents the average percentage of correct answers over the last fifty cycles, averaged over ten trials for each curve.

3.3 The effect of population size

For serial implementations of classifier systems, population size directly affects the speed of the system; computational cost is proportional to the product of the average message-list size and the population size. However, classifier systems are inherently parallel, and *CFS demonstrates that the speed of the system can be independent of population size in a parallel implementation. Until recently, researchers believed that the larger the population, the better. However, Goldberg (1985) has theorized that there is an optimal population size for pure GA. If the population is too small, the system converges too quickly and does not process enough schemata. If the population is too large,

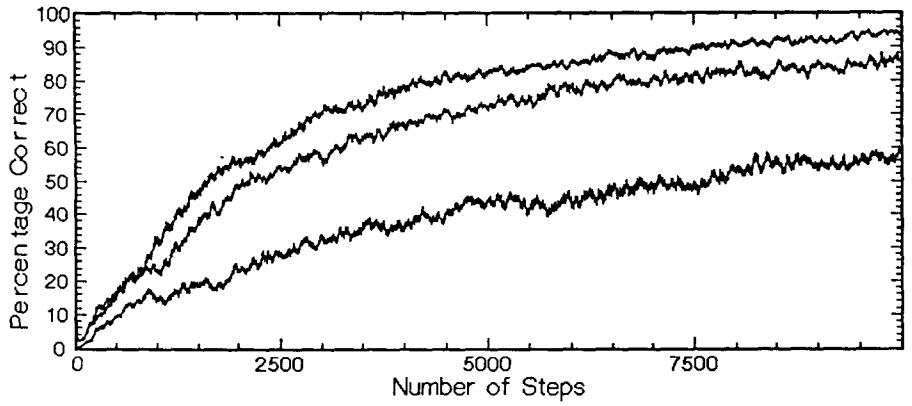

Figure 2. Learning the alphabet over 10000 steps. Population sizes are 50 (lowest curve), 500 (middle), and 8000 (upper).

the waiting times for effective crossovers are too long and there is insufficient juxtaposition of building blocks prior to convergence. Goldberg's theory is based on counting unique expected schemata derived from strings of a given length and populations of a given size. It is not clear that Goldberg's theory can be directly applied to classifier systems, because such systems work on several problems at once and because their discovery algorithms include more than GA. To the extent that we can apply Goldberg's theory, the empirical results that we see tend to contradict the theory.

The task of letter sequence prediction uses nine-bit messages; with two conditions and an action, the chromosome used in the GA contains 27 bits. According to Goldberg's theory, the optimal population size for a string length of 27 is 77. However, the theory does not account for subproblems that must be solved. In this task domain, there are as many subproblems as there are letters in the sequence being predicted. If we assume independence of the classifiers working on each subproblem, then the optimal population size for "abcd" should be 308 classifiers, for "mississippi" it should be 847, and for the alphabet it should be 2002. If they are not independent (which is very likely the case), the optimal size should be smaller.

Figure 1 shows the results for "abcd," which indicate that performance continues to increase as population size increases well past the theoretical optimum size of 308. Larger populations lead to faster learning and ultimately to better performance. The same results were seen for "mississippi," a problem with some ambiguity. Figure 2 shows the same results for the alphabet, a longer problem.

These results are based on using GA, as well as the cover detector and cover effector operators (CDO/CEO). However, Figure 3 shows the learning behavior of GA alone on the alphabet. As with the other results, performance increases as population size increases, well past the theoretical optimum of 2002. This appears to directly contradict Goldberg's theory of optimal population size.

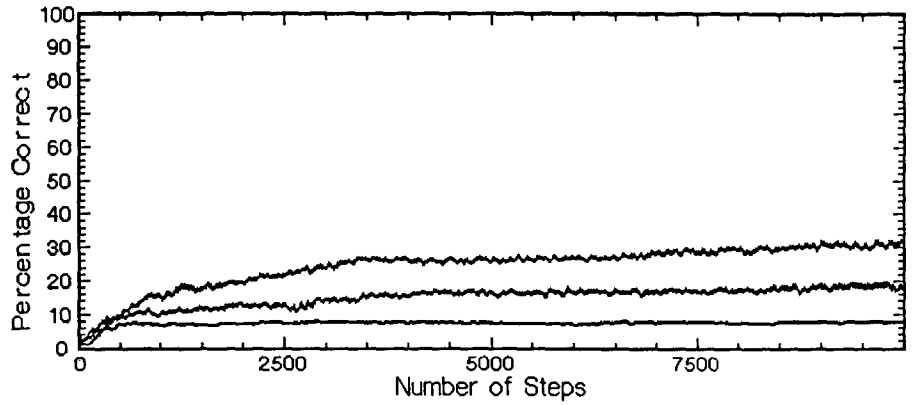

Figure 3. Learning the alphabet over 10000 steps, only using GA. Population sizes are 500 (lowest curve), 2000 (middle), and 8000 (upper).

In reviewing these results, Goldberg has suggested modifying his theory to allow multiple nonoverlapping populations each handling some different "concept" and to require k copies of some good schema (not the one copy he assumed in his original work). With these modifications, our results tend to support his basic contention that there is a fundamental tradeoff between having a copy of a schema of a particular length and having to generate that schema some time later.

To summarize, the theory of population size developed for pure GA does not apply directly to classifier systems. It must be modified to take into account the multiple subproblems being worked on and their interactions. It must also be modified to deal with the effects of other discovery heuristics (particularly CDO and CEO) on the GA. Our empirical results indicate that increasing population size increases performance, although with diminishing marginal returns after some point.

3.4 The contribution of genetic algorithms

As can be seen by comparing Figures 2 and 3, the genetic algorithm alone does not contribute as much to the system's learning behavior as one might expect. In particular, using a population of 8000 classifiers on the alphabet task, the system achieves a performance of over 90% when the cover detector, cover effector, and GA are all used, but it reaches only about 30% performance when the GA is used in isolation.

In order to separate the effects of GA from those of the cover operators (CDO/CEO), we ran a series of experiments with one or the other mechanism disabled. Figure 4 shows the results for a population of 8000 on the sequence "abcd." Similar curves were seen for the "mississippi" task, and Figure 5 shows the results for the alphabet task.

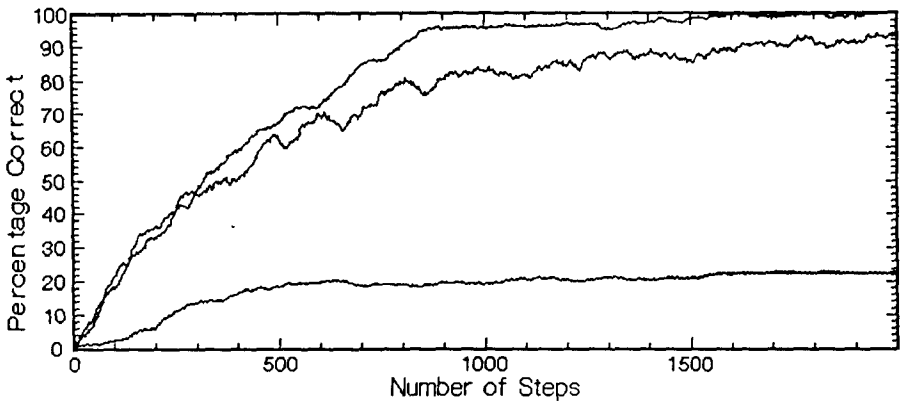

Figure 4. Learning the letter sequence "abcd." The lowest curve is for GA only, the middle is for cover operators only, and the upper is for both mechanisms.

For each task the system does best when both the CDO/CEO and GA are used together. Also, the combination CDO/CEO does better than the GA on each task, even though the former involves a random walk through a portion of the solution space. For the easiest ("abcd") task, CDO/CEO does almost as well as using both CDO/CEO and GA, whereas for the harder alphabet problem, CDO/CEO does noticeably worse than when both mechanisms are used. On the other hand, GA in isolation leads to about the same or slightly better performance on the harder tasks than it does on the easy task. What can account for these results? Our hypothesis consists of three parts:

1. Using GA alone, the system rapidly reaches a plateau at a relatively low performance level because it *prematurely converges* (Booker, 1982; De Jong, 1975) to a population with many copies of just a few kinds of classifiers. Once converged, the crossover operator is no longer effective, so learning stops.
2. The cover operators act to enrich the gene pool for the GA, so that the crossover operator again can carry out an efficient search. Thus when CDO/CEO and GA are used together, the system performs best.
3. The system can improve its performance using just CDO/CEO because the tasks are relatively easy. For the "abcd" task, CDO/CEO alone can do quite well, but for the two harder tasks, CDO/CEO does much worse than the CDO/CEO and GA together, and it does only somewhat better than GA alone.

To test this hypothesis, we ran a series of experiments to test the effects of convergence on performance. An initial population was constructed by generating one random classifier and then making 7999 copies of it. This artificial situation represents the ultimate converged population. If our hypothesis were true, we would expect the following:

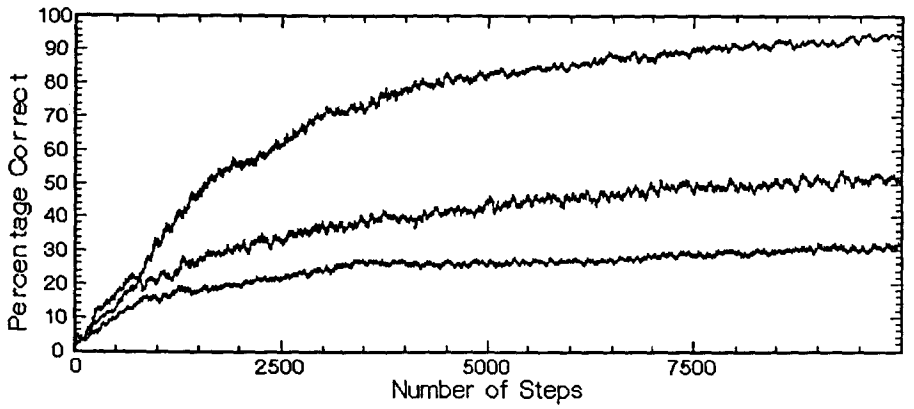

Figure 5. Learning the alphabet. The lowest curve is for GA only, the middle is for cover operators only, and the upper is for both mechanisms.

1. The system should perform poorly when only GA is used, since there is no variance in the population for the crossover operator to exploit.
2. Using just CDO/CEO, the system should perform about the same when started with the fully converged population as it does when started with a random initial population, since CDO/CEO does not make use of the variance in a population.
3. Using both CDO/CEO and GA, the system should be able to learn starting from a converged population, though perhaps not as well as it does when started with a random initial population. The GA can begin to contribute to the learning only after CDO/CEO injects variance into the gene pool.

In fact our results were somewhat mixed. The first two predictions were confirmed for all three tasks ("abcd," "mississippi," and the alphabet). The third prediction was only partially confirmed. This suggests that the convergence hypothesis depends in some way on the complexity of the task and does not hold for all cases.

In addition to premature convergence, another factor that may influence the amount of learning contributed by GA is task representation, and one important aspect of representation involves the choice of an evaluation function. One evaluation function we tried for the letter sequence prediction domain payed a fixed reward (positive payoff) for a correct prediction and a fixed punishment (negative payoff) for any other answer. With this evaluation function the search space is mostly flat and negative, with just a few positive spikes or mesas for correct rules. This type of search space is not easily searched by the GA (or any algorithm), since most instances provide no information about how close the rule is to a solution.

In the experiments described in this paper, a partial payoff scheme was used in which the payoff increased for each bit of the prediction that was correct and

there was no punishment (the payoff is zero) for getting all bits wrong. This evaluation function produces a space that has many hills and valleys, so that partial solutions can act as *building blocks* which the GA can combine to find complete solutions. Thus, selecting a good representation is the crucial first step toward making GA an effective learning mechanism for classifier systems.

A third factor that can influence GA techniques is the evolution of species. Booker (1982) has shown that GA can perform better in classifier systems that must solve multiple subproblems when a *restricted mating* mechanism is used to promote and exploit the formation of "species" of classifiers.[1] The hypothesized reason for this behavior is that restricted mating improves the effectiveness of the crossover operator, since the offspring of a crossover between parents from two species is less likely to be a useful classifier.

In the experiments described so far, nothing was done to explicitly encourage or discourage speciation, but there are several possible ways to promote it. One approach is to arbitrarily divide the population into several fixed-sized subpopulations, thus allowing the subpopulations to evolve in relative isolation. Although these subpopulations may find different niches and evolve into different species, there are no guarantees. Also, there is no *a priori* way of knowing how many species are required to solve a problem, hence there is no way of knowing how to initially subdivide the population.

A second approach involves applying a sharing function to the fitness measure. This technique identifies the local density of a species and adjusts fitness according to the number and closeness of its neighbors. This approach was suggested by Holland (1981) and Booker (1982), applied successfully by Wilson (1985), and extended by Goldberg and Richardson (1987). A third approach, suggested by Booker (1982), biases parent selection toward classifiers that bid in response to the same messages. This approach seems to capture at least some of what happens in natural speciation. In the next section we describe some preliminary experiments that use this form of restricted mating to promote speciation in the context of tasks that require rule chains.

In summary, our results indicate there is an interesting (but not fully understood) synergy between GA and the cover operators that make them quite powerful when used together. Since these results may depend on the particular task domain or representation, we believe similar empirical studies should be carried out in other domains, and that a theoretical foundation for the interaction of these mechanisms should be developed.

3.5 Sequences of coupled classifiers

As described in Section 2.4, sequences of coupled classifiers, or *classifier chains*, are necessary to let classifier systems implement arbitrary networks and perform a variety of computations. In the context of letter sequence prediction, classifier chains provide a way to implement a short-term memory, which enables the system to make predictions not determined directly by detector messages.

[1] A "species" in a classifier system may be thought of as a set of classifiers that all solve the same subproblem.

For example, consider the sequence "yhwh" when the system's window has a size of one. When the letter seen is an "h," the system cannot use just the detector message to predict the next letter, since sometimes the next letter is "y" and sometimes "w." To solve this problem, the system must include classifiers that implement a short-term memory. For instance, the following classifiers (in interpreted form) predict part of the "yhwh" sequence perfectly:

```
[1] If Detect ( y ) then Predict  ( h ).
[2] If Detect ( y ) then Remember ( y ).
[3] If Detect ( h )  and Remember ( y )  then Predict ( w ).
```

When the current letter is "y," classifiers 1 and 2 both post messages: the message produced by classifier 1 activates the effector and the system predicts an "h," whereas that produced by classifier 2 is an "internal memory" message. On the next step, the message list contains a detector message that indicates the current letter ("h") *and* it contains the messages produced during the previous step. Classifier 3 is then activated by matching the detector message and the message posted by classifier 2. Thus, classifier 2 is *coupled* to classifier 3, and this coupling lets the system correctly predict a "w" *two* steps after a "y." A similar set of classifiers could be added to predict a "y" two steps after a "w."

3.5.1 Problems with the bucket brigade algorithm

In learning classifier systems, it is important that credit (strength) be allocated to classifiers that lead to good performance by the system as a whole. In the context of sequences of coupled classifiers, this means that a classifier like the second one in the above example, which "sets the stage" for a classifier that actually makes a prediction, must be given credit for enabling the system to make that prediction correctly. The *bucket brigade algorithm* (BBA) serves this function.

One problem with the BBA occurs when a sequence of coupled classifiers contains classifiers which have conditions that match messages produced by sources other than their predecessors in the sequence. In general, the strengths of the stage-setting classifiers will be much lower than those of effector-activating classifiers (Riolo, 1987a). For example, in the set of classifiers described above, classifier 3 has one condition that matches a detector message and a second that matches the message produced by its predecessor, classifier 2. Using the standard BBA, the strength of classifier 2 will be one-half that of classifier 3 (since 3 would pay $\frac{1}{2}$ of its bid to 2 and $\frac{1}{2}$ for the detector message). Because the GA tends to replace low-strength classifiers, an uneven distribution of strength between classifiers in sequences means that the stage-setting classifiers will tend to be eliminated.

To avoid this problem, the BBA in CFS-C was altered so that when a classifier matches detector messages and messages produced by classifiers, the entire bid is paid to the classifiers and nothing is paid for the detector messages. However, if a classifier uses only messages from detectors, that classifier's entire bid is paid for those messages.

A second problem with the traditional BBA is the misallocation of credit to stage-setting classifiers that are active when a payoff is received from the environment. Using the standard BBA, *all* classifiers that are active when a reward is received from the environment have that reward added to their strength. For example, when classifier 1 correctly predicts the occurrence of an "h," both it *and* classifier 2 (which is active on the same step) would receive the reward for a correct prediction. Thus classifier 2 would receive credit for a prediction it did not make directly or indirectly.

To avoid this problem, the BBA was modified so that *only* those classifiers that produce effector-activating messages receive payoffs from the environment. Classifiers like number 2 will only receive strength from their successors in a chain.

A third problem results from the application of taxes. Since taxes are applied to every classifier, the strengths of classifiers in a chain fall off exponentially as they are further removed from the payoffs earned by executing that chain. For example, at a tax rate of 0.025, the fixed-point strength of a stage-setting classifier is only 80% that of the classifier it activates. For the experiments described in this section, taxes were kept low (HeadTax = 0.001, BidTax = 0.02, and ProdTax = 0.0125 for one message).

3.5.2 Experiments with coupled classifiers: Maintaining chains

For a classifier system to establish and maintain a stable solution to a task like that of predicting the sequence "yhwh" using a window of size one, the system must not only be able to *discover* (create) a set of classifiers to solve the problem, it must also be able to *maintain* such a set once it has been discovered. The rest of this section describes experiments carried out to show that the CFS-C system can, with some modifications, both discover and maintain solutions to tasks that require coupled classifiers.

A series of experiments was carried out to determine if the system could maintain stable performance when it is *started* with classifiers that solve the "yhwh" task. For each run, the system was started with four copies of each of the classifiers described earlier, along with 76 randomly generated classifiers. The initial strength of the random classifiers was $\frac{1}{4}$ the strength of the classifiers in the solution set. The GA and cover operators were applied to generate new classifiers and replace others, while the BBA (modified as described earlier) was used to allocate strength. The message list size was four, and the system performance was tracked for 7000 steps.

After trying a number of tax rates, discovery operator application rates, and so on, the only way we found to consistently maintain a stable solution set was to modify the system in three ways.

Limit the maximum number of copies of each type of classifier. Stage-setting rules tend to have lower strengths than effector-activating classifiers because (a) tax rates were greater than zero and (b) stage-setters sometimes are not paid for producing messages (e.g., when the classifiers they are coupled to do not win the bidding competition on the next step). With repeated applications of the GA, offspring of higher strength effector-activating classifiers tend to spread

through the population, replacing the lower strength stage-setting classifiers. Multiple copies of classifiers all producing the same effector message make it difficult for the lower strength stage-setters to win the bidding competition, which in turn causes their strength to fall further. When the last copy of a stage-setting classifier is lost, the coupled sequence is disrupted. To alleviate this problem, the classifier replacement algorithm was changed so that when a new classifier is generated, if the classifier list already contains a fixed number of copies (e.g., two) of that classifier, the new classifier just replaces the copy that has the lowest strength.

Reserve a portion of the message list for "internal memory" messages. Since different classifiers can match the same messages and produce the same messages, after repeated applications of the GA the populations were found to contain many copies of effector-activating classifiers that were not identical but that behaved identically. This again kept the stage-setting classifiers from winning the competition to post messages. To overcome this problem, the message list size was increased so that a total of eight messages could be posted by classifiers, but one-half of the message list was reserved for messages produced by stage-setting classifiers.

Restrict application of the crossover operator to co-bidding parents. The standard way to apply the GA in *CFS or CFS-C is to use strength as the measure of a classifier's "fitness." Since any classifier might be mated with any other, many offspring tend to be useless; e.g., when one parent responds to one situation and the other responds to a different situation, the offspring may not match any situation. When a population contains stage-setting classifiers, the offspring of crossovers between classifiers active at different steps often disrupt the proper execution of coupled classifiers. Although the BBA generally eliminates these problem classifiers after a few trials, the allocation of trials to incorrect coupled classifiers also leads to a lowering of strength in the *correctly* coupled classifiers (e.g., when a stage-setter is not paid for a message it produced because an incorrect classifier wins the competition.) To reduce the production of problem-causing offspring, the GA was modified so that classifier *bids* were used to pick parents. In effect this is a *restricted mating* scheme (Booker, 1982) in which only classifiers that bid during the same step are allowed to produce offspring modified by crossover.

After the CFS-C system was changed to include these modifications, stable solutions to the "yhwh" task were maintained in the test runs. For a fairly broad range of parameter settings (different taxes, GA rates, etc.), performance remained above 90%.

3.5.3 Experiments with coupled classifiers: Discovering chains

Next, experiments were run to determine whether or not CFS-C could discover a solution to the "yhwh" task when started with a *random* set of 100 classifiers, with a window size of one. Figure 6 shows the average performance for two sets of runs. In one set, only the cover operators and the GA were used to generate new classifiers. In the other set, the triggered chaining operator

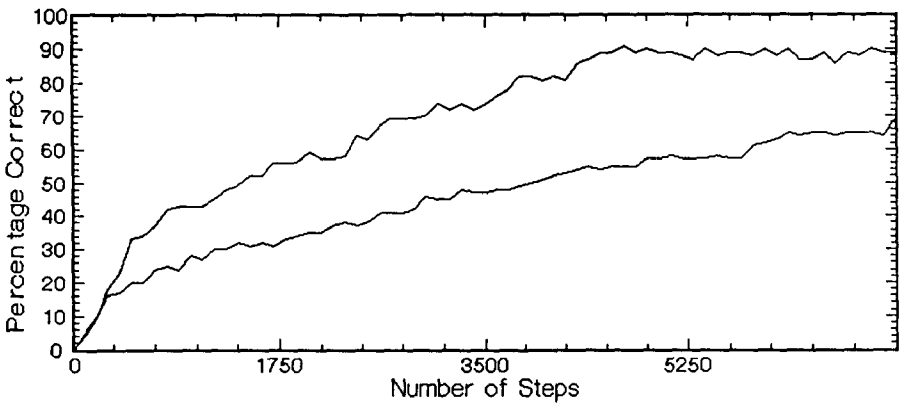

Figure 6. Learning the letter sequence "yhwh" over 7000 steps. The upper curve is with the triggered chaining operator and the lower curve is without this operator.

(or TCO) was also applied once every ten steps. All other parameters were the same.

The TCO significantly improves learning in this task domain. The average performance in the last 1750 steps without the TCO was 62.4 (standard deviation 14), whereas with the TCO it was 88.5 (standard deviation 9). Without the TCO, seven runs reached a performance of about 75% whereas three reached about 50%. With the TCO, all runs had a performance greater than 73%; six were greater than 92% and two greater than 83%. Examination confirmed that coupled classifiers were created in the eight runs with the TCO that obtained prediction rates greater than 80%.

The ten runs using the TCO were repeated without reserving a portion of the message list for internal memory messages. For these runs, the average performance was 67.1%; in only three runs did chains evolve and become established. Thus, reserving a portion of the message list for internal memory messages dramatically improves the ability of CFS-C to evolve and maintain solutions that require coupled classifiers.

Table 1 shows the results of limiting the number of duplicate classifiers to different maximum copies. Although the high variance across runs makes it impossible to reach a definite conclusion, these results suggest that limiting the number of duplicate classifiers to two or four may be best for promoting the discovery and maintenance of coupled classifiers.

As a further test of the modified CFS-C system, it was run on a subset of the "mississippi" sequence, "mississi," which requires the use of coupled classifiers to solve the problem even when the window has length four. Figure 7 shows the learning curves that result when only the GA and CDO/CEO are used, as well as when these were combined with the TCO. Each run was continued for 16,000 steps, starting with 200 random classifiers, and each curve is an average of five runs.

Table 1. Effects of limiting duplicate classifiers.

MAXIMUM COPIES	PERFORMANCE (% CORRECT)	NUMBER OF RUNS WITH CHAINS
1	62.1	1
2	88.5	8
4	83.2	6
6	80.2	6

The TCO leads to slightly improved learning and performance for the "mississi" task. The average performance without the TCO was 61.5% (standard deviation 7), whereas the average with the TCO was 71.8 (standard deviation 5). Although examination of the classifiers did show the existence of coupled classifiers in the TCO runs, those classifiers did not solve the difficult part of the task, namely what to predict when the most recent letters are "issi." From the experiments carried out so far, it is not clear why the system was not able to solve this problem. Perhaps longer runs or runs with more classifiers are required. It also seems likely that the existence of both repeating letters and overlapping ambiguous sequences makes this an especially difficult problem.

In summary, the experiments show that classifier systems can learn to perform simple tasks that require formation of sequences of coupled classifiers, and that this can be accomplished using the TCO in addition to the GA and other discovery algorithms. However, even when using the TCO, coupled classifiers were not always created, and in more difficult tasks this method did not lead to the classifiers necessary to solve the task. More research will be necessary to determine how to form coupled classifiers more consistently.

In our tests, we made a number of changes to CFS-C in order to facilitate maintenance of classifiers chains. Although some of these changes were rather heavy-handed ways to limit the dominance of the classifier and message lists by higher strength effector-activating classifiers, it may be possible to use more dynamic methods to achieve the same ends. For example, "reward sharing" (Booker, 1982; Wilson, 1985) and "crowding" schemes (Booker, 1982; De Jong, 1975) may accomplish the same effects.

3.6 Convergence and representation

Classifier systems converge on a useful set of classifiers after some number of task examples. In other words, after the classifiers are evolved to handle portions of the task, they begin to replicate in the population. As this process takes place, alleles are lost, some loci become fixed on one allele, and crossover becomes less effective because its search space is reduced. With a population that has totally converged on a single classifier type, crossover has no effect at all. The rate of convergence is not well understood, even though premature convergence has long been known to be a major problem with GA (De Jong, 1975). In addition to inhibiting crossover, premature convergence leads to ineffective use of the limited message list; multiple copies of the same classifier

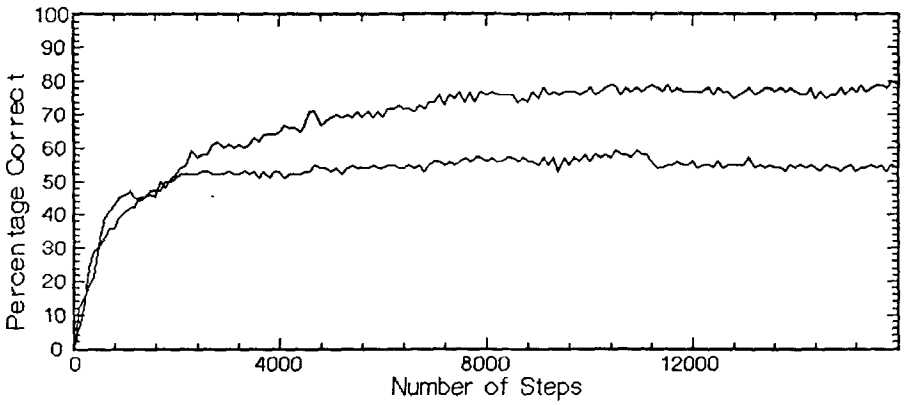

Figure 7. Learning the letter sequence "mississi" over 16000 steps. The upper curve is with the triggered chaining operator and the lower curve is without this operator.

fill the message list and prevent other classifiers from expressing themselves. Premature convergence is analogous to getting trapped in a local maximum in hill climbing. Both mutation and the cover effector operator help avoid local maxima, and thus decrease the chance of premature convergence.

In order to understand the effects of convergence rate, we have developed an approximate hyperplane convergence measure for classifier systems. There are two major problems associated with developing this measure. First, wildcards in classifiers make it impossible to do a static analysis of hyperplane convergence. In order to correctly account for each wildcard allele, a convergence measure must know what messages were available to match that allele at that instant in time. This kind of dynamic analysis is too computationally expensive to allow continuous monitoring of convergence. To avoid this problem, our measure is static and ignores wildcards; thus it only approximates the actual hyperplane convergence of the population.

The second problem involves the lack of explicitly identified species in the population. A correct measure of hyperplane convergence would be relative to a species, but lacking information about species, our hyperplane convergence measure holds across the entire population. The formula is

$$C = \frac{1}{m} \sum_{i=1}^{m} \frac{|N_{0_i} - N_{1_i}|}{(N_{0_i} + N_{1_i})}.$$

where m is the number of loci in the whole classifier. N_{0_i} is the number of 0 alleles at locus i, and N_{1_i} is the number of 1 alleles at locus i. C ranges between zero for total divergence and one for total convergence.

Figure 8 shows the behavior of this approximate measure of hyperplane convergence. The lowest curve is for a random population with only cover operators; it starts with nearly total divergence and remains that way. The

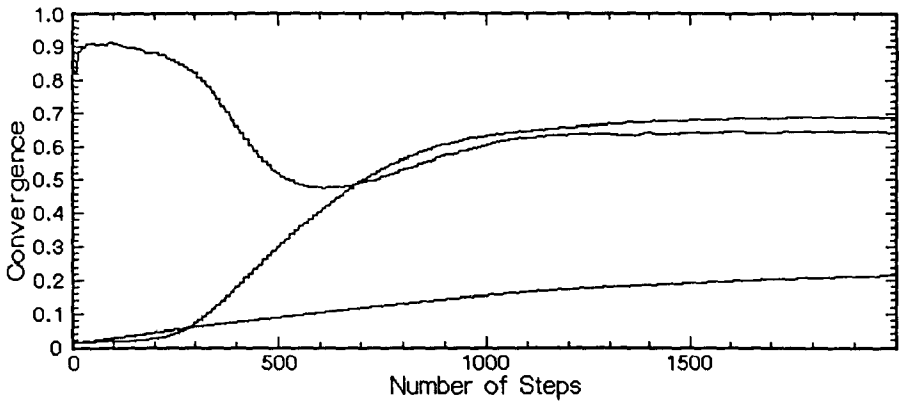

Figure 8. Examples of convergence. The lowest curve is for a random population with only cover operators, the middle curve is for a random population with both discovery mechanisms, the upper curve is for an identical initial population with both mechanisms.

middle curve is for a random population with both GA and CDO/CEO; it starts diverged and converges to about 0.68 after 1000 cycles. The highest curve is for a population of identical classifiers with both mechanisms; it starts essentially converged (not totally, because we are ignoring wildcards) and diverges to approximately the same C level as the middle curve.

Why is an effective convergence measure so important? First, looking at the empirical behavior of a convergence measure over time may aid in the development of a theory of convergence. The second reason relates to improving the efficacy of task representations. For example, one can learn the "abcd" sequence using only two bits for the letter representation instead of five. Figure 9 illustrates the gain if one uses the simpler representation on this particular problem. The upper curve shows the learning behavior using two bits to represent letters, whereas the lower curve shows the behavior using five bits. Learning is about twice as fast when the more appropriate representation is used. However, there is no *a priori* way of knowing which particular problems from the task domain will be encountered.

Shaefer's (1987) ARGOT suggests one approach to solving this problem. The system adapts its representation during learning in order to seek the most effective encoding. ARGOT uses a convergence measure to control the shrinkage and growth of its representation. However, this approach has so far only been applied to pure genetic algorithms. In order to introduce an ARGOT-like mechanism into classifier systems, we must promote speciation and find an effective convergence measure. The resulting system could then take advantage of convergence and could improve its problem representation in the process of learning.

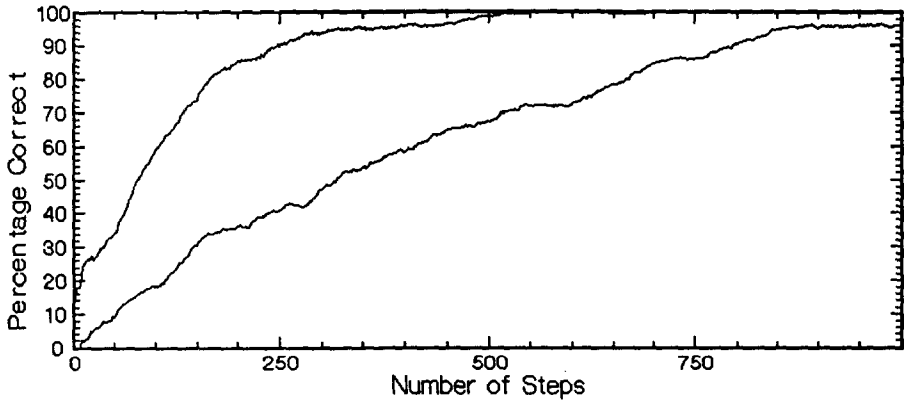

Figure 9. Learning the letter sequence "abcd" over 1000 steps. The lower curve is for a nine-bit message representation and the upper curve is for a six-bit representation.

4. Summary

Classifier systems are a promising approach to developing machines that can learn. Two general-purpose classifier systems, CFS-C and *CFS, have been implemented and validated against each other using a letter sequence prediction task. The inherent parallelism of classifier systems and genetic algorithms has been realized and demonstrated on a massively parallel computer with *CFS. The speed of this system is independent of the classifier population size, allowing exploration of larger task domains than have previously been practical.

We have examined several known problems with classifier systems. On the issue of population size, we found evidence that increasing population size increases learning, contrary to existing theory. On the issue of GA impact on classifier systems, we found evidence that a random walk through part of the solution space dominates GA for simple problems but that, in general, both mechanisms together work far better than either in isolation. On the issue of sequences of rules, there has been a long-standing concern because rule chains almost never evolve naturally in classifier systems. We found evidence that a triggered chaining operator introduces such chains and that the bucket brigade algorithm preserves them. We have also demonstrated an approximate hyperplane convergence measure, but a more precise measure is needed. With a good convergence measure, we could introduce ARGOT-like mechanisms, which appear to offer significant advantages.

In addition, we need simpler mechanisms for controlling overgeneralization. The tax system helps control overgeneralization, but it is complex and causes other problems. For example, the head tax has a deleterious effect over the long term if the environment is changing. An alternative way to eliminate non-participatory classifiers would be to change the replacement selection algorithm from selection based on fitness to a 'least recently used' algorithm.

Much work remains in both the theory and practice of classifier systems before we can fully realize their potential. The encouraging sign is that no fundamental problems have been found and proposed solutions are being investigated for each of the problems we have described.

Acknowledgements

This work was supported in part by Grant IRI 86-10225 from the National Science Foundation. We would like to thank John Holland and Arthur Burks for support of this work in general and in particular for travel funds that they provided, Stewart Wilson for his suggestions of alternative mechanisms, and Craig Shaefer and Steve Smith for their insights on hyperplane convergence measures and the application of ARGOT to classifier systems. We would also like to thank Bob Axelrod, Michael Cohen, Dave Davis, Danny Hillis, Jill Mesirov, Carl Simon, Michael Savageau, Dave Waltz, and the editors, David Goldberg, John Holland, and Pat Langley, for their helpful comments and suggestions.

References

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

Booker, L. B., Goldberg, D. E., & Holland, J. H. (in press). Classifier systems and genetic algorithms. *Artificial Intelligence.*

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.

De Jong, K. A. (1988). Learning with genetic algorithms: An overview. *Machine Learning, 3,* 121–138.

Goldberg, D. E. (1985). *Optimal initial population size for binary-coded genetic algorithms* (TCGA Report No. 85001). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Goldberg, D. E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (pp. 41–49). Cambridge, MA: Lawrence Erlbaum.

Hillis, W. D. (1985). *The connection machine.* Cambridge, MA: MIT Press.

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

Holland, J. H. (1981). *Genetic algorithms and adaptation* (Technical Report No. 34). Ann Arbor: University of Michigan, Department of Computer and Communication Sciences.

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

Holland, J. H. (1985b). *A mathematical framework for studying learning in classifier systems* (Research Memo RIS No. 25). Cambridge, MA: Rowland Institute for Science.

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., & Burks, A. W. (1987). *Adaptive computing system capable of learning and discovery.* United States patent application.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1986). *Induction: Processes of inference, learning, and discovery.* Cambridge, MA: MIT Press.

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.

Riolo, R. L. (1986a). *CFS-C: A package of domain independent subroutines for implementing classifier systems in arbitrary, user-defined environments* (Technical Report). Ann Arbor: University of Michigan, Division of Computer Science and Engineering, Logic of Computers Group.

Riolo, R. L. (1986b). *LETSEQ: An implementation of the CFS-C classifier system in a task domain that involves learning to predict letter sequences* (Technical Report). Ann Arbor: University of Michigan, Division of Computer Science and Engineering, Logic of Computers Group.

Riolo, R. L. (1987a). Bucket brigade performance: I. Long sequences of classifiers. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (pp. 184–195). Cambridge, MA: Lawrence Erlbaum.

Riolo, R. L. (1987b). Bucket brigade performance: II. Default hierarchies. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (pp. 196–201). Cambridge, MA: Lawrence Erlbaum.

Robertson, G. G. (1987). Parallel implementation of genetic algorithms in a classifier system. In L. Davis (Ed.), *Genetic algorithms and simulated annealing.* London: Pitman Press.

Robertson, G. G., & Riolo, R. L. (1987). *A tale of two classifier systems* (Technical Report RL87-6). Cambridge, MA: Thinking Machines Corporation.

Shaefer, C. G. (1987). The ARGOT strategy: Adaptive representation genetic optimizer technique. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (pp. 50–58). Cambridge, MA: Lawrence Erlbaum.

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

Wilson, S. W. (1987). Classifier systems and the animat problem. *Machine Learning, 2*, 199–228.