Machine Learning 3: 225–245, 1988
© 1988 Kluwer Academic Publishers – Manufactured in The Netherlands

Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms

JOHN J. GREFENSTETTE (GREFENSTETTE@AIC.NRL.NAVY.MIL)
Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, DC 20375-5000, U.S.A.

(Received: December 21, 1987)
(Revised: July 15, 1988)

Keywords: Credit assignment, rule discovery, genetic algorithms, classifier systems, bucket brigade algorithm

Abstract. In rule discovery systems, learning often proceeds by first assessing the quality of the system's current rules and then modifying rules based on that assessment. This paper addresses the credit assignment problem that arises when long sequences of rules fire between successive external rewards. The focus is on the kinds of rule assessment schemes which have been proposed for rule discovery systems that use genetic algorithms as the primary rule modification strategy. Two distinct approaches to rule learning with genetic algorithms have been previously reported, each approach offering a useful solution to a different level of the credit assignment problem. We describe a system, called RUDI, that exploits both approaches. We present analytic and experimental results that support the hypothesis that multiple levels of credit assignment can improve the performance of rule learning systems based on genetic algorithms.

1. Introduction

This paper addresses the problem of learning heuristic rules for the application of known operators. Systems for learning heuristic rules often proceed in two phases. Existing rules are first assessed in a problem-solving context; some of these rules are then modified in the hope of improving the performance of the system on similar tasks. The rule assessment phase usually gives rise to a *credit assignment* problem (Minsky, 1961): If a sequence of rules fires before the system solves a particular problem, how can credit or blame be accurately assigned to early rules that set the stage for the final result? For example, in a chess-playing program the decision that immediately precedes checkmate is not usually to blame for the final outcome. Rather, some rule that fired earlier in the game may be responsible for the fatal sequence of moves, but it is often difficult to identify the responsible rule.

If a complete, tractable domain theory is available, then analytical learning techniques (Mitchell, Keller, & Kedar-Cabelli, 1986) might be applied. If one can assume that an optimal solution path is known (or can be produced by the problem-solving module), then an analysis of solution traces can provide positive and negative instances of rule applications (Sleeman, Langley, & Mitchell, 1982; Langley, 1983; Mitchell, Utgoff, & Banerji, 1983). If very little background knowledge is available and the problem environment provides a natural measure for the quality of outcomes, it is appropriate to view the problem of learning as a search for high-performance knowledge structures, and to explore the capabilities of genetic algorithms (Holland, 1975) to perform the search. In this paper we explore the latter approach.

Holland, Holyoak, Nisbett, and Thagard (1986) describe a complete theory of inductive systems based on competitive principles of economics and population genetics. Our aim is more modest, namely, to examine a few of the possible computational mechanisms that might be used to realize that theory. Two rather distinct approaches to rule learning using genetic algorithms have been developed. In one approach, illustrated by the LS-1 system (Smith, 1980, 1983), each knowledge structure in the population represents a production-system program stated as a list of rules. As a result of applying the knowledge structure to the problem-solving task, the entire program is assigned a fitness measure, which is used to control the selection of structures for reproduction. Genetic search operators are applied to the selected structures, producing a new population of production-system programs. In practice, LS-1 has successfully learned maze tasks and played successful poker against a version of Waterman's (1970) poker player.

Another approach is taken in *classifier systems* (Holland & Reitman, 1978; Holland, 1986; Wilson, 1987a), in which the genetic operators are applied to single production rules, or *classifiers.* Each rule is assigned a *strength* measure that indicates the utility of the rule to the system's goal of obtaining external reward. New rules are discovered by genetic operators applied to existing rules that have been selected on the basis of strength. The newly created rules must successfully compete with established rules in order to survive. Given an appropriate representation for the rules (Holland, 1986), the theory of genetic search predicts that the successful new rules will be plausible variants of their precursers, and that classifier systems can discover significant sets of cooperative rules. The classifier system approach has been implemented in several successful learning systems (Booker, 1982; Goldberg, 1983; Wilson, 1985; Zhou, 1987).

The relative merits of these two approaches is a topic of active debate in the genetic algorithm research community (De Jong, 1987). This paper presents the view that each approach offers an interesting solution to a distinct aspect of the credit assignment problem. Classifier systems assign credit to individual rules and, through the genetic algorithm, to parts of rules (Holland, 1986). LS-1 assigns credit to lists of rules and, through the genetic algorithm, to clusters of rules that serve as building blocks for high-performance knowledge structures. The remainder of the paper presents an explicit comparison of various credit assignment techniques used in genetic learning systems, together with a new rule learning system, RUDI, that tries to exploit each of these techniques to its best advantage and an experimental comparison of the alternative methods under discussion.

2. Credit assignment in the LS-1 system

Smith (1983) has described a system called LS-1 that employs a genetic algorithm to search for high-performance knowledge structures, each structure consisting of a list of rules. The system maintains a population of knowledge structures that evolves over time as a result of its experiences. A fitness measure for each knowledge structure is obtained by observing the performance of a problem-solving module that uses the given rules to solve a number of tasks from the problem domain. Once each knowledge structure in the population has been evaluated, a new population of structures is formed in two steps.

First, a *selection* procedure chooses structures for replication by a stochastic process that ensures that the expected number of offspring associated with a given structure is proportional to the structure's observed performance relative to the other structures in the current population. As a result, high-performance structures may be chosen several times for replication and low-performance structures may not be chosen at all. In the absence of any other mechanisms, this selective pressure would cause the best-performing structures in the initial knowledge base to occupy a larger and larger proportion of the knowledge base over time.

The second step produces new plausible knowledge structures by *recombining* pairs of selected structures using idealized genetic operators. The primary genetic operator is *crossover*, which combines two parent structures to form two similar offspring.[1] Crossover operates at multiple levels of granularity in LS-1 by exchanging segments of the string or list representations of the parents. For example, if the parents are represented by the lists

`( A, B, C, D, E )` and `( a, b, c, d, e )`,

then crossover at the rule level might produce the offspring

`( A, B, c, d, e )` and `( a, b, C, D, E )`.

The primary effects of crossover are to create new combinations of rules (such as { `b, C` } above) and to test established combinations (such as { `A, B` }) in new contexts. A secondary role is to create new rules from existing rules. This is accomplished in LS-1 by allowing crossover at the level of individual symbols between rule boundaries. Thus, crossover can produce the offspring

`( A, B′, c, d, e )` and `( a, b′, C, D, E )`,

where rule `B′` and rule `b′` each contains parts of rule `B` and parts of rule `b`. Given the representation language for rules, these new components can be viewed as plausible variants of rules that are associated with high-performance parents.

[1]In contrast to earlier studies of evolutionary programming (Fogel, Owens, & Walsh, 1966), random mutation plays a minor role in genetic algorithms as a background operator that maintains the reachability of all points in the search space.

The combined effect of performance-biased reproduction and crossover is a sophisticated form of adaptive search through the space of knowledge structures. This search process quickly identifies combinations of rules that are associated with high performance and propagates them through the knowledge base. In fact, this effect has been quantified in the *schema theorem* for genetic algorithms, established by Holland (1975) and extended to encompass LS-1's operator set by Smith (1980). The schema theorem states that the number of structures in the knowledge base that share a given subset of components can be expected to increase or decrease over time at a rate proportional to the observed performance of the subset. That is, genetic algorithms allocate an appropriate level of credit to all subsets of components appearing in the knowledge base *simultaneously*, without the explicit computation of the underlying statistics. This property, known as the *implicit parallelism* of genetic algorithms, means that, even though utility is explicitly computed only for entire sets of rules, credit assignment in LS-1 operates implicitly at the level of much smaller groups of rules.

The implicit credit assignment in LS-1 is especially effective if related rules are clustered, since rules that are physically close together on the list representing the knowledge structure stand a good chance of being inherited as a group. For this reason, the system includes an *inversion* operator (Holland, 1975) that reverses a randomly chosen contiguous set of rules on a given knowledge structure, thereby altering the probability that the affected rule combinations will be disrupted by crossover. Smith (1983) found that a moderate rate of inversion produced slight performance improvements in LS-1, but he also found that random inversion is too weak to effectively identify new representations. One explanation is that, faced with the huge space of rule permutations, the random changes in representation provided by inversion cannot be expected to make much difference before the population converges under the selection pressure of the genetic algorithm. The system needs a more effective way to identify subsets of rules that form cooperative groups and to cluster these groups within each knowledge structure. We will refer to the problem of identifying related rules as the *clustering* problem for LS-1. We present one solution to this problem in Section 4.

Another problem with credit assignment in LS-1 is that it has no direct way to accumulate and exploit knowledge about the utility of individual rules. Schaffer and Grefenstette (1985) have described an extension of Smith's system, called LS-2, that allows a richer form of credit assignment. The new system's critic provides a set of performance measures that rate each rule list's ability to solve independent aspects of the performance task. Subpopulations of rule lists are selected on the basis of each performance measure, providing the equivalent of environmental niches in which specialists can evolve. Recombination in LS-2 disregards the subpopulation boundaries, so that offspring can inherit specialized rules from parents with different areas of specialization. LS-2 performed well on a multi-class discrimination problem in the domain of human gait analysis. Section 4 presents a more direct way of using individual rule utilities within the LS-1 framework, but first we describe how such rule utilities can be computed using techniques developed for classifier systems.

3. Credit assignment in classifier systems

A primary difference between LS-1 and classifier systems is that the latter assign a utility measure called *strength* to individual rules rather than to entire production-system programs. As the result of genetic operators being applied to high-strength rules, a classifier system is expected to gradually refine its set of rules to adapt to its task environment. Just as the genetic algorithm in LS-1 implicitly operates on small groups of rules based on explicit fitness measures assigned to entire rule sets, the genetic algorithm in classifier systems implicitly exploits information about components of rules based on the strength assigned to individual rules (Holland et al., 1986). In this paper, we focus on the explicit credit assignment mechanisms that operate on the rule level. Two different strength-updating schemes have appeared in the classifier systems literature, but they have not been explicitly compared before now. As shown below, each scheme measures a distinct form of utility. In Section 4, we show how to incorporate both forms of utility in a genetic learning system.

3.1 A simple classifier-system model

We focus our discussion on simplified classifier systems that learn heuristic control rules for applying a set of known operators to a set of states. Each rule in these systems has the form:

```
IF   the current state is in the set $S_i$,
THEN apply operator $O_i$.
```

The left-hand side of each rule contains patterns which can be matched against detector messages that indicate the observed state.[2] On each step, every rule that matches the current state makes a *bid* where $bid = b \times strength$. The constant b is called the *bid-ratio*, where it is assumed that $0 < b < 1$. In this simplified model, a single rule is selected to *fire* from among the matched rules using a probability distribution defined by the bids. That is, the probability of rule R_i with bid bid_i firing is bid_i/bid_M, where bid_M is the sum of all bids made by the rules in the match set. When a rule fires, the operator associated with the rule is applied to the current state. The result becomes the new current state. If no rule matches the current state, a randomly chosen operator is used to produce the next state. Certain states are associated with *external reward*. The goal of the system is to learn a set of rules that maximizes the external reward obtained over all possible starting states in the state space.

Our goal in defining this simple classifier-system model is to shed some light on appropriate interpretations of a rule's strength under alternative strength-updating schemes. In order to facilitate our analysis, we have restricted our model in comparison to the general classifier-system model (Holland, 1986; Riolo, 1986). In that model, rules may have several conditions, which may match messages from other rules as well as the current state. The bid may depend on other factors such as the generality of the rule, strengths may be

[2]A detailed discussion of state representation and matching is beyond the scope of this paper. Holland (1986) discusses appropriate pattern languages for classifier systems and Booker (1982) discusses issues in matching.

reduced by a variety of taxes, and more than one rule may fire on a single step. Holland et al. (1986) discuss the justifications and uses for these additional mechanisms in a general-purpose learning system. Despite the assumptions we will make to aid our analysis, we believe that the essential insights gained from this basic model apply as well to the general model, as we discuss below.

3.2 The profit-sharing plan

We first consider a strength-updating scheme that we call the *Profit-Sharing Plan (PSP)*, a modified version of the method described by Holland and Reitman (1978). In this scheme, problem solving is divided into *episodes* delimited by the receipt of external reward. A rule is said to be *active* during a given episode if it wins a bidding competition during any step of the episode. At the end of episode t, the PSP updates the strength $S_i(t)$ of each active rule R_i according to the rule

$$S_i(t+1) \;=\; S_i(t) \;-\; bS_i(t) \;+\; bp(t), \tag{1}$$

where $p(t)$ refers to the external reward obtained at the end of the episode. That is, when external reward is obtained, one bid is collected from each active rule, and each active rule is given a fraction of the external reward.[3] Notice that this rule may be rewritten as

$$\Delta\ S_i(t) \;=\; b(p(t) - S_i(t)),$$

which reveals its similarity to the well-known Widrow-Hoff rule (Widrow & Hoff, 1960).

Consider the effect of the PSP on a given rule R_i. It follows from equation (1) that

$$S_i(t) \;=\; (1-b)^t S_i(0) \;+\; b\sum_{i=1}^{t}(1-b)^{t-i}p(i-1), \tag{2}$$

where t ranges over the episodes during which rule R_i is active. That is, $S_i(t)$ is essentially a recency-weighted average of the external reward $p(t)$, with $(1-b)$ serving as the exponential decay factor. If b is sufficiently small, then $S(t)$ hovers around the average of recent values for $p(t)$. If the external reward $p(t)$ is a constant, p^*, then S_i converges to an equilibrium value S_i^* given by

$$S_i^* \;=\; \lim_{t\to\infty} S_i(t) \;=\; \lim_{t\to\infty}[(\ (1-b)^t S_i(0) \;+\; b\sum_{i=1}^{t}(1-b)^{t-i}p^*\ ] \;=\; p^*. \tag{3}$$

Under constant payoff, it follows from equation (2) that the PSP reduces the error $E_i(t) \;=\; p^* \;-\; S_i(t)$ at a rate given by

[3]There are many reasonable variations. For example, one might collect one bid for each time the active rule fired during the episode. Holland and Reitman (1978) attenuate rewards so that rules firing earlier are usually rewarded less than those firing closer to the end of an episode. The particular form of the PSP described here was chosen in order to clarify the relationship between reward schemes that wait until external reward is obtained and incremental schemes such as the bucket brigade algorithm in the next section.

$$\begin{aligned}\Delta E_i &= E_i(t+1) - E_i(t) \\ &= S_i(t) - S_i(t+1) \\ &= -b(p^* - S_i(t)) \\ &= -bE_i(t). \end{aligned} \tag{4}$$

That is, each change in strength reduces the difference between the current strength and the equilibrium strength by a factor of b. For example, suppose that $p^* = 500, b = 0.1$, and $S_i(t) = 100$. Then $S_i(t+1) = 100 - 10 + 50 = 140$. Note that the error $p^* - S_i$ decreases from 400 to 360, or by ten percent.

In summary, under conditions of relatively constant payoff, the strength of each rule under the PSP rapidly converges to an equilibrium strength which predicts the level of reward that will be received at the end of the episode. One potential limitation of the PSP is that it depends on the assumption that the episodes delimited by successive external rewards correspond to appropriate periods over which credit assignment takes place. That is, if rule R fires at step τ and the next external reward is obtained at step $\tau + k$, then R does not share in any external reward obtained after $\tau + k$ (unless it fires again). The choice of episode thus appears to be an important consideration. For example, if the PSP were used in a chess-playing system, the proper choice for the episode is probably an entire game. If any external reward is issued at intermediate points, situations may arise in which stage-setting rules are not given credit for their consequences that occur after the next external reward. We now consider an alternative credit assignment method that avoids these assumptions about episodes.

3.3 The bucket brigade model

Most recent classifier systems (Booker, 1982; Goldberg, 1983; Riolo, 1986; Wilson, 1987a) have adopted a distributed, incremental credit assignment scheme, called the ***bucket brigade algorithm (BBA)***, that is based on individual transactions among rules (Holland, 1985). In our simplified classifier-system model, if rule R_i fires during step τ and rule R_j fires during step $\tau + 1$, then the BBA updates the strength S_i of rule R_i according to the rule

$$S_i(\tau + 1) = S_i(\tau) - bS_i(\tau) + bS_j(\tau). \tag{5}$$

This is identical to equation (1) except that the episode index t has been replaced by the step index τ and the external payoff $p(t)$ has been replaced by the strength S_j of rule R_j. The first change means that the BBA might update the strength of a rule more than once during a given episode.[4]

The second change results in a fundamental difference between the PSP and the BBA. Whereas strength in the PSP predicts the expected external payoff that is received at the end of the episode in which the rule fires, strength in the

[4]This is not a significant difference between the BBA and the PSP. Recall that we could have defined the PSP to update the strength of a rule by an amount proportional to the number of times the rule fired during the episode.

BBA predicts the expected internal payoff that the rule receives in the next step. Consider two rules, R_i and R_j, that are coupled in the sense that each firing of R_i is immediately followed by the firing of R_j, and assume that this pair fires at most once in any episode. Then we have

$$S_i(t) = (1-b)^t S_i(0) + \sum_{i=1}^{t} b(1-b)^{t-i} S_j(i-1), \tag{6}$$

where t ranges over the episodes during which both rules are active.[5] In other words, the strength of R_i tracks the strength of R_j. If S_j converges to a constant value S_j^*, then S_i converges as well:

$$S_i^* = \lim_{t\to\infty} S_i(t) = \lim_{t\to\infty} [\, (1-b)^t S_i(0) + b\sum_{i=1}^{t}(1-b)^{t-i} S_j(i-1)\,] = S_j^*. \tag{7}$$

In fact, the same reasoning that leads to equation (4) shows that S_i converges to S_j^* (the *internal payoff* for R_i) at the same rate as a rule's strength converges to a prediction of *external reward* under the PSP. This analysis extends to any *chain* of rules, e.g., a collection of rules $< R_1, R_2, \ldots, R_n >$ that fire in sequence, and in which the only external reward (if any) is received by R_n.

Proposition 1. Under equilibrium conditions, all rules in any chain have equal strength under the bucket brigade algorithm.

Proof. At equilibrium, the bid paid by each rule is exactly matched by the bid received from the next rule or the reward received from the environment. Let $< R_1, R_2, \ldots, R_n >$ be a chain of rules. Let S^* be the equilibrium strength of rule R_n. For each i, $1 \leq i < n$, the bid paid by R_i is bS_i^* and the payoff received by R_i is bS_{i+1}^*. At equilibrium, $bS_i^* = bS_{i+1}^*$. Therefore, $S_i^* = S^*$ for $1 \leq i \leq n$. ■

Of course, equilibrium conditions are unlikely to occur in practice, because the levels of external reward received by the final rule in a chain may be variable, or because a rule may appear in more than one chain. Under nonequilibrium conditions, Proposition 1 must be modified to the extent that the strengths of the rules in the chain are described by equation (6) rather than by equation (7). In any case, if the expected reward received by a chain is relatively consistent, the strengths of the rules in the chain tend to converge to a common level.

The BBA is closely related to the temporal-difference methods described by Sutton (1988). An important difference is that the particular methods analyzed by Sutton assign predictive measures (strength) to *states* rather than to *rules*. Given that the number of states is usually too large to explicitly maintain such statistics, this appears to be an advantage for the BBA, but the relationships among these methods should be explored in further detail.

[5]Goldberg (1983) presents a similar analysis of the bucket brigade algorithm.

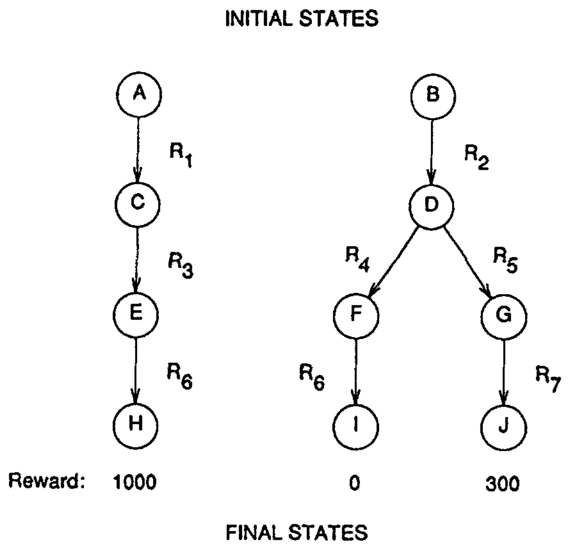

Figure 1. State space for experiment comparing the profit-sharing plan (PSP) and the bucket brigade algorithm (BBA).

3.4 Comparing the PSP and the BBA

In many cases, the PSP and the BBA lead to identical results, but it is not hard to devise scenarios in which the expected internal reward differs from the expected external reward. Consider the example in Figure 1, which shows ten states, including the initial states A and B and the final states H, I, and J. The external rewards generated at states H, I, and J are 1000, 0, and 300, respectively. To make the example concrete, we might imagine an end-game in Othello, in which state H corresponds to a win by a wide margin (measured by piece-advantage over the opponent), state I is a loss, and state J is a marginal win.[6] Each episode begins in a randomly selected initial state and ends when a final state is reached. The key feature of this example is that some rule (R_6) can fire in two possible states, one firing leading to high reward and the other leading to low reward. For example, R_6 might be a rule that suggests a certain move that leads to victory when it fires in state E, but leads to a clear loss when it fires in state F. It should be clear that this combination of circumstances is fairly likely to arise in any large problem in which rules match more than one state.

Classifier systems (with the genetic learning operators disabled) were run on this example, using either the PSP or the BBA to update rule strengths.

[6]The "external reward" might also take into account the costs of firing the rules. For example, states H and J might both be equally desirable but rule R_2 might consume more internal resources than R_1.

Table 1. The effects of different strength-updating schemes.

RULE	PSP-STRENGTH	BBA-STRENGTH
R_1	1000	648
R_2	299	567
R_3	1000	645
R_4	4	644
R_5	300	300
R_6	999	531
R_7	300	300

All initial strengths were set to 100, the bid ratio b was set to 0.1, and 1000 episodes ending in external reward were performed (i.e., 3000 steps of each classifier system). The resulting rule strengths are shown in Table 1.

The example shows that the PSP-strength for each rule has converged toward the expected external reward received when that rule fires. For example, the PSP-strength of R_4 converges toward zero. Once the strength of R_4 decreases sufficiently, R_6 is effectively prevented from firing in state F. As a result, the PSP-strength of R_6 converges to 1000. The PSP-strength of R_2 reflects the fact that the maximum possible reward (300) is always obtained when starting from initial state B. In contrast, the BBA-strength of R_2 and R_4 are inflated with respect to expected external payoff by the presence of R_6 in the chain.

This example supports Westerdale's (1985) characterization of the BBA as a *subgoal reward* scheme. That is, the condition part of each rule represents a subgoal, a set of states to which that rule can be applied. The BBA assigns credit to a rule (say R_4) based on achieving the subgoal associated with the successful rule R_6, not on whether R_6 can in fact deliver external reward once state F is reached. Although Westerdale's analysis appears correct, the heuristic value of subgoal rewards is unclear. For example, the high BBA-strength of R_4 does not indicate that its subgoal (state D) occurs in a sequence leading to high external reward, but only that its subgoal leads to another subgoal (associated with R_6) that (in other cases) leads to high external reward. Carrying the analysis one step further, it is not clear that rule R_2 should be rewarded for achieving the subgoal associated with R_4. There appears to be a decoupling between the internal payoff (reflected by the BBA-strength) and the external payoff that can be obtained when R_4 fires (in this case, zero).

This example suggests that the PSP generally gives more reliable guidance for conflict resolution than does the BBA. For example, R_4 has a higher BBA-strength than R_5, even though R_5 leads to external reward and R_4 does not. As a result, in the classifier system using the BBA, state I is visited more often than state J even though the payoff is less for I than for J. For example, state I might represent the loss of game, but the BBA provides no help in identifying the decision (i.e., the choice of R_4 over R_5) that is responsible for the poor outcome. On the other hand, since PSP-strength predicts the expected level of external reward, it makes sense to perform conflict resolution based on this measure. The effect of these alternative schemes is reflected in the average external payoff obtained by each system over the period of 1000 episodes: the BBA system obtained an average reward of 526 out of a maximum average external reward of 650, whereas the PSP obtained an average reward of 628.

This example also suggests that care must be taken when basing rule discovery operations on BBA-strength. For example, R_4 has higher BBA-strength than R_7, but R_7 leads to external reward and R_4 does not. Given this observation, it seems questionable whether a genetic algorithm should prefer the reproduction and modification of R_4 over R_7, or the deletion of R_7 over R_4. In contrast, it seems sensible for a rule discovery algorithm to prefer high PSP-strength rules for reproduction and modification, or to prefer low PSP-strength rules for deletion.

On the other hand, BBA-strength does measure dynamic coupling among rules; i.e., rules that appear in frequently occurring chains (such as $R_2 \rightarrow R_4 \rightarrow R_6$) tend to have similar levels of BBA-strength, as predicted by Proposition 1. In contrast, the PSP is not helpful in identifying sequentially related groups of rules. In fact, rules that tend to fire in succession may have highly dissimilar levels of PSP-strength. For example, if the external reward at state I were 300, then the final strength of R_6 would be between 1000 and 300, but the equilibrium strengths of R_1 and R_3 would still be 1000. In Section 4, we will show how to use this feature of the BBA to solve the clustering problem for LS-1.

Given the differences between the PSP and the BBA, it is important to address the relative costs of these two schemes. It is clear that the BBA is a more *local* algorithm than the PSP, in that it modifies strength at each step on the basis of a local transaction between two rules. Since no detailed trace of past rule firings is necessary, the BBA seems especially well-suited to systems in which parallel rule firing is appropriate (Holland, 1986). In contrast, the PSP relies on the assumption that there is a single active chain of rules during any one episode, and may not generalize well to systems that allow parallel rule firing. The BBA updates at least as many strengths per episode as the PSP, and so the BBA takes at least as much time as the PSP on sequential machines; however, the BBA is better suited than the PSP for implementation on massively parallel processors (Robertson, 1987). The PSP requires slightly more memory than the BBA to record the active rules. In short, the impact of the differences between the PSP and the BBA depends on the overall architecture of the learning system, the nature of the hardware on which it is implemented, and the accuracy required of the credit assignment method.

It is important to assess the role of our simplifying assumptions in the example analyzed above. Since the general classifier-system model is still evolving, we will only address a few of the features that have commonly appeared in such systems. Most classifier systems allow, but do not require, conditions to match messages from other rules as well as the external state. This would have no effect on the example, since the specified situation might arise among rules that do not happen to depend on internal messages. Likewise, many classifier systems allow, but do not require, multiple rules to fire in parallel. The example illustrates a situation where only one rule happens to fire at a time.

In many classifier systems, the bid-ratio is a function of the generality of the pattern in the condition part of the rule. Making the bid proportional to the rule generality could be easily accommodated without changing the essential features of the example: imagine that the states represent equivalence classes of actual states such that the cardinality of E and F is half that of the other classes; then all rules have equal generality in terms of the number of states matched, and their bid-ratios would be constant assuming an appropriate representation of the state-detector messages. Researchers have proposed other mechanisms that alter the bidding mechanism, or the distribution of payoff when multiple rules fire in parallel, or that periodically tax away portions of each rule's strength (Holland, 1985; Riolo, 1986). A fuller understanding of the effects of such mechanisms on our analysis must await further research. However, it seems reasonable to conjecture that the essential observation – that strength under the BBA serves to predict internal payoff but not necessarily external rewards – depends on the nature of the BBA itself rather than the particular classifier-system model we have presented.

In summary, the BBA and the PSP appear to provide complementary information about the utility of rules in a competition-based rule discovery system. Given that the BBA seems to have some implementation advantages, especially in systems that allow multiple rules to fire in parallel, one attractive direction for further research is to investigate the forms of strength-updating rules that enable a local algorithm like the BBA to provide the kind of credit assignment provided by the PSP.

4. RUDI: A multilevel system for credit assignment

We may summarize the results of the previous sections as follows: credit assignment in LS-1 operates on the level of rule groups, but it is hampered by its lack of knowledge about the performance of individual rules and by the inability of random inversion to create meaningful clusters of rules. The PSP and the BBA provides complementary utility information about individual rules. PSP-strength provides a more accurate estimate of the utility of a rule in terms of its expected external reward. In contrast, BBA-strength indicates the dynamic associations among rules, with rules that fire in sequence converging to similar levels. This section describes one way to exploit all of these credit assignment techniques in a single system.

4.1 The problem-solving level

RUDI (for RUle DIscovery) is a system that combines features of LS-1 and classifier systems, as shown in Figure 2. The problem-solving level consists of a simplified classifier system, as described in Section 3.1. No genetic operators are applied at this level, but the problem solver does update both the PSP-strength and the BBA-strength of active rules. Since the first measure provides an estimate of expected external reward, it is used for conflict resolution during problem solving. The latter measure is used by the learning level to cluster cooperative sets of rules, as described below. The problem solver reports to

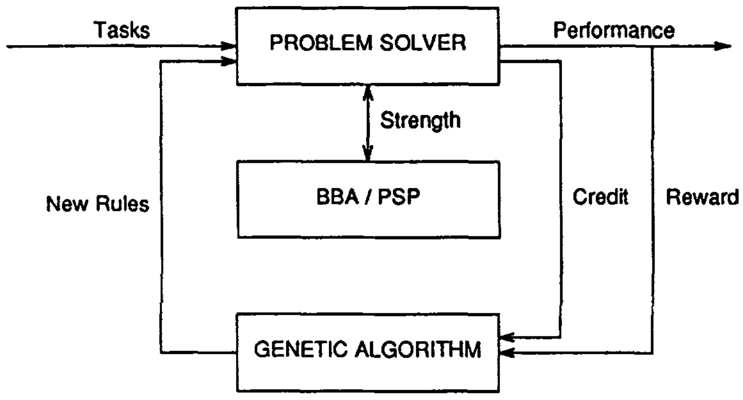

Figure 2. The control structure of RUDI, a multilevel genetic learning system.

the learning level, providing the average external reward received by each rule set during the solution of a set of tasks from the problem domain, as well as the updated rule strengths.

4.2 The learning level

As in LS-1, the genetic learning algorithm in RUDI operates on a population of knowledge structures, each of which is represented by a list of rules. The overall performance of the knowledge structures controls the selection of these structures for reproduction. Modified structures are formed by applying crossover to the selected structures, as in LS-1, except that each rule in the offspring inherits the strengths associated with the corresponding rule in the parent structure.[7]

RUDI differs from LS-1 in that the strengths of individual rules influence the physical representation of the knowledge structures, making it more likely that useful combinations of rules survive and propagate throughout the knowledge base. This is accomplished by a heuristic form of inversion called *clustering*. This process is applied just prior to crossover and involves two steps:

(1) Sorting the rules within the knowledge structure on the basis of their BBA-strength.
(2) Treating the knowledge structure as a cycle and shifting the sorted rules a random number of positions along the structure.

The shift phase is necessary in order to avoid a bias against certain distributions of building blocks. For example, it allows (but does not guarantee) an offspring to inherit all of the high BBA-strength rules from both parents. Figure 3 shows an example of clustering a rule set.

[7]In the case of a new rule created by crossing in the middle of two rules, the new rule is assigned the average strength of the two parent rules.

BEFORE CLUSTERING:

Rule:	R_1	R_2	R_3	R_4	R_5
Strength:	225	50	250	30	300

(1) SORT BY BBA-STRENGTH:

Rule:	R_5	R_3	R_1	R_2	R_4
Strength:	300	250	225	50	30

(2) CIRCULAR SHIFT:

Rule:	R_4	R_5	R_3	R_1	R_2
Strength:	30	300	250	225	50

Figure 3. An example of the cluster operator's effects on a rule list.

As shown in Section 3.3, rules that frequently occur in the same problem-solving chain will tend to have similar levels of BBA-strength. Clustering moves such rules closer together on the knowledge structure, and since crossover takes contiguous groups of rules from each parent, rules occurring frequently in the same chain will tend to be inherited as a group. This heuristic provides a way to form meaningful clusters of cooperative rules that was missing from LS-1.

5. An experimental comparison of methods for credit assignment

This section describes experiments that compare RUDI's combination of credit assignment mechanisms with those mechanisms in isolation. These initial experiments were run on generic problem-solving tasks designed to capture the essential features of the credit assignment problem. The goal was to learn heuristic rules for the application of a fixed set of operators so as to maximize the expected reward over all possible starting states. Space permits the detailed discussion of only one set of experiments, but similar results have been achieved on other problems with various state-space topologies and distributions of rewards to final states.[8]

[8]Experiments with six other state spaces have been performed, varying the total number of states, the depth of the search space, and the rewards associated with final states. The problem presented here represents one of the more challenging problems studied. Further systematic comparisons are required before we can precisely delimit the power of the various credit assignment methods.

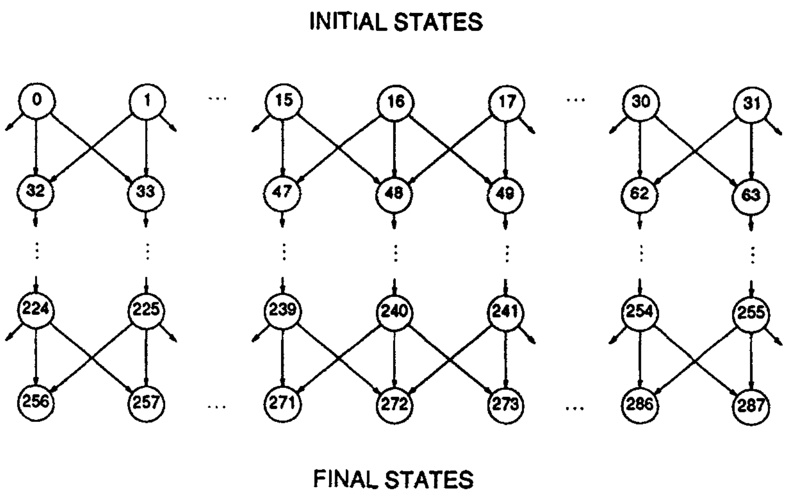

Figure 4. The state space used in experiments on credit assignment.

5.1 The problem domain

The test problem consisted of a state space containing 288 states arranged in a rectangular array with 9 rows and 32 columns, as shown in Figure 4. There were 32 initial states and 32 final states, and each traversal of the space required eight decision steps. The non-final states were identified by an eight-bit feature vector, the first three bits indicating the current row and the next five bits indicating the current column. Three operators – LEFT, STRAIGHT, and RIGHT – mapped each state to its successors. For example, LEFT(16) = 47, STRAIGHT(16) = 48 and RIGHT(16) = 49. The state space was cylindrical, so that LEFT(0) = 63.

The rewards associated with the final states ranged from 0 to 1000, with an average of 250. As shown in Table 2, the rewards were distributed around two hills of high scores, with payoff valleys between. The distribution of external rewards to the final states was chosen so that, in order to gain maximum reward, the system would have to discover correct heuristic rules for the early stages of each task. For example, to obtain maximum reward from starting state 0 (at the upper left corner of the state space), it was necessary to apply the operator RIGHT for at least seven of the eight steps in the task (in order to reach maximum reward at state 263 or 264).

5.2 Representation for rules

As with any learning method, the knowledge representation employed by a genetic learning system constitutes an important bias. In the learning systems tested here, the left-hand side of each rule contains a pattern that is matched

Table 2. Distribution of external rewards for the state space in Figure 4.

STATE	REWARD	STATE	REWARD
256	0	272	0
257	0	273	0
258	50	274	50
259	75	275	75
260	125	276	125
261	250	277	250
262	500	278	500
263	1000	279	1000
264	1000	280	1000
265	500	281	500
266	250	282	250
267	125	283	125
268	75	284	75
269	50	285	50
270	0	286	0
271	0	287	0

against the eight-bit feature vector describing the current state, using the symbol # to indicate that the value of the corresponding feature is irrelevant.[9] The right-hand side of each rule indicates a single operator, using a fixed mapping from integers to operators. For example, the rule

$$00000\#\#1 \rightarrow 2$$

represents the heuristic

```
IF   the current state is in the set {1,3,5,7},
THEN apply operator STRAIGHT.
```

Given this representation, there are a total of 19,683 distinct rules, with 768 maximally specific rules (each matching a single state). Recognizing that it is generally infeasible to consider all possible rules (or even all maximally specific rules), we limited the rule memory of each system to a maximum of 128 rules.

5.3 Experimental results

A series of experiments was performed to test the effects of various credit assignment methods on the problem domain. Although there are ways to inject available knowledge into genetic learning systems (Grefenstette, 1987), for the purposes of these experiments we decided to start all the learning systems with randomly generated knowledge structures. Since genetic learning systems are stochastic processes, all figures show the average results of five independent runs.

[9]Holland (1975) discusses the possibility of learning new features, but we do not pursue that approach here.

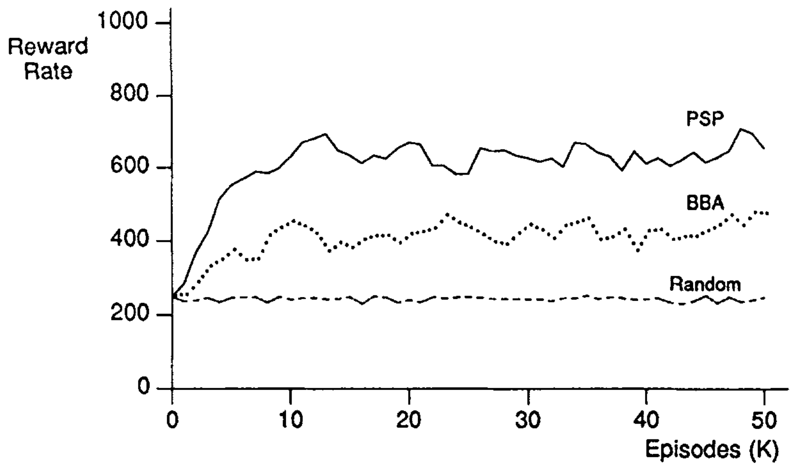

Figure 5. Performance profiles of systems using different credit assignment methods.

Figure 5 shows performance profiles for a random walk algorithm and for two simple classifier systems that used either the PSP or the BBA for conflict resolution and for rule reproduction. Each data point in Figures 5 and 6 reflects the average reward obtained during the previous 1000 episodes. The system using PSP clearly dominated the system using BBA, but both left much room for improvement, since the maximum possible reward per episode was 1000. These results are not conclusive, since they depend on many parameter settings (e.g., bid-ratio, population size, and frequency of genetic operators) for which there is little available tuning experience (Riolo, 1986). However, they are consistent with previous studies in which classifier systems have had difficulty in performing successful credit assignment over chains of similar length unless the BBA is augmented by specially designed *bridge classifiers* (Riolo, 1987).

Figure 6 shows the performance of three LS-1 style systems that used different forms of credit assignment. The system denoted LS-1.0 used random conflict resolution and no clustering at the learning level. The one denoted LS-1.5 used PSP for conflict resolution, but also performed no clustering at the learning level. Finally, RUDI used PSP-strength for conflict resolution and BBA-strength for clustering at the learning level, as described above. Each system maintained a knowledge base of 50 knowledge structures, each consisting of 64 rules along with their associated strengths. Each structure was evaluated by using its heuristic rules in 20 tasks, each task starting at a randomly chosen initial state and ending at a final state. The 20 tasks were chosen independently for each rule set evaluation. Each run consisted of 2500 rule set evaluations (50 generations).[10]

[10]For the LS-1 style systems, the genetic algorithm at the learning level used parameter settings consistent with previous studies (De Jong, 1975; Grefenstette, 1986): population size = 50; crossover rate = 0.6; mutation rate = 0.001; generation gap = 1.0; scaling window

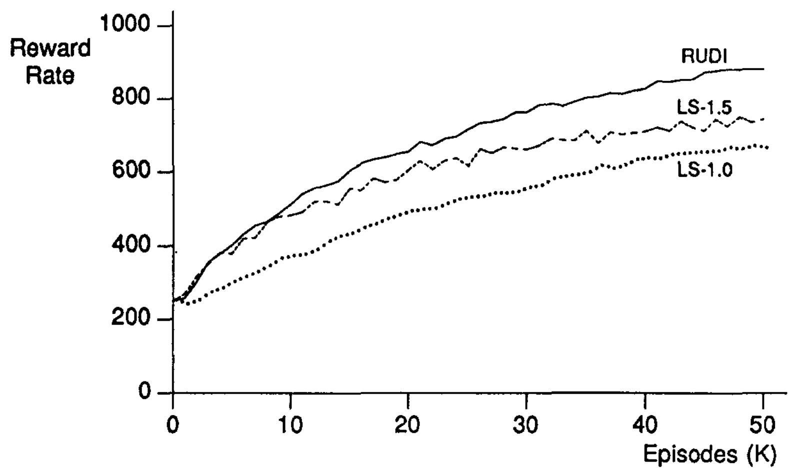

Figure 6. Performance profiles of LS-1 style learning systems.

After each run, the rule set with the highest evaluation in the final population was subjected to a final evaluation consisting of 1000 reward episodes. The average results for all runs are shown in Table 3. The clear advantage in RUDI's performance over the other systems supports the hypothesis that multiple levels of credit assignment can lead to better performance of rule discovery systems based on genetic algorithms.

6. Conclusions

This paper has examined issues of credit assignment in rule-learning systems based on genetic algorithms. The classifier-systems approach and the LS-1 approach each provides useful mechanisms for assigning credit. RUDI represents a new method of reconciling these two approaches, using the bucket brigade algorithm from classifier systems to solve the clustering problem in LS-1. The new system demonstrates the benefits of exploiting multiple levels of credit assignment, but further testing is needed to delimit the class of problems for which this approach is most valuable.

An important topic for further research involves the development of local strength-updating schemes like the BBA that predict levels of external reward to the degree achieved by the PSP scheme. Such schemes would be especially useful in systems that allow parallel rule firing (Holland, 1986). Wilson (1987b) has described another approach to the credit assignment problem in classifier systems. He proposes a hierarchical form of classifier system in which strength is passed only among modules that operate at the same level of abstraction. For example, a system that learns restaurant behavior might have high-level

= 5. We include these parameter settings for completeness; genetic algorithms are robust over a fairly broad range of control parameter settings.

Table 3. Average performance of final rule sets from Figures 5 and 6.

LEARNING SYSTEM	EXTERNAL REWARD
OPTIMAL	1000
RUDI	943
LS-1.5	791
LS-1.0	724
PSP	632
BBA	468
RANDOM	250

modules such as ENTER-RESTAURANT, GET-A-TABLE, and EAT appearing in one reinforcement chain, while lower-level modules, such as FIND-DOOR and OPEN-DOOR, form a distinct reinforcement chain. Wilson proposes a hierarchical BBA that would manage the interface between chains at various levels, while keeping the chains at each level relatively short. This seems to be a promising approach that deserves further attention.

The credit assignment methods analyzed in Section 3 are independent of the rule modification processes that might use this information, and could be used in learning systems that use rule modification mechanisms other than genetic operators. For example, Langley (1983) has described SAGE, an adaptive production system for procedural learning that starts with overly general rules and gradually modifies them through discrimination operators. Each rule in SAGE has an associated strength, which is increased each time the rule is recreated through the discrimination process. Conflicts between rules are settled by selecting the highest-strength rule. The system assumes that procedures for identifying incorrect rule applications are available or that an optimal solution path can be generated to serve as a model in learning process. The addition of a more sophisticated credit assignment method and conflict resolution mechanism, similar to the PSP or the BBA, might permit the extension of SAGE to problems in which the direct detection of incorrect rule applications is impossible and optimal solutions paths cannot be easily generated.

Acknowledgements

I want to thank Lashon Booker and Ken De Jong for many stimulating discussions on this topic, and Rick Riolo for providing a copy of his CFS-C software. I also wish to thank the editors for many thoughtful comments that improved the presentation of this work.

References

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

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

De Jong, K. (1987). On using genetic algorithms to search program spaces. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (pp. 210–216). Cambridge, MA: Lawrence Erlbaum.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution.* New York: John Wiley.

Goldberg, D. E. (1983). *Computer-aided gas pipeline operation using genetic algorithms and rule learning.* Doctoral dissertation, Department of Civil Engineering, University of Michigan, Ann Arbor.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, 16,* 122–128.

Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algorithms. In L. Davis (Ed.), *Genetic algorithms and simulated annealing.* London: Pitman Press.

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

Holland, J. H. (1985). 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. (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., 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.

Langley, P. (1983). Learning effective search heuristics. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 419–421). Karlsruhe, West Germany: Morgan Kaufmann.

Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers* (pp. 8–30).

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1,* 47–80.

Mitchell, T. M., Utgoff, P. E., & Banerji, R. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Riolo, R. L. (1986). *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. (1987). 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.

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.

Schaffer, J. D., & Grefenstette, J. J. (1985). Multi-objective learning via genetic algorithms. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 593-595). Los Angeles, CA: Morgan Kaufmann.

Sleeman, D., Langley, P., & Mitchell, T. M. (1982). Learning from solution paths: An approach to the credit assignment problem. *AI Magazine*, *3*, 48–52.

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

Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 422–425). Karlsruhe, West Germany: Morgan Kaufmann.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

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

Westerdale, T. H. (1985). The bucket brigade is not genetic. *Proceedings of the First International Conference on Genetic Algorithms and Their Applications* (pp. 45–59). Pittsburgh, PA: Lawrence Erlbaum.

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. *1960 WESTCON Convention Record, Part IV* (pp. 96–104).

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

Wilson, S. W. (1987b). Hierarchical credit allocation in a classifier system. In L. Davis (Ed.), *Genetic algorithms and simulated annealing.* London: Pitman Press.

Zhou, H. H. (1987). *CSM: A genetic classifier system with memory for learning by analogy.* Doctoral dissertation, Department of Computer Science, Vanderbilt University, Nashville, TN.