

Do We Really Need to Estimate Rule Utilities in Classifier Systems?

Lashon B. Booker

The MITRE Corporation
1820 Dolley Madison Blvd
McLean, VA 22102-3481, USA
booker@mitre.org

Abstract. Classifier systems have traditionally used explicit measures of utility (strength, predicted payoff, accuracy, etc.) to quantify the performance and fitness of classifier rules. Much of the effort in designing and implementing classifier systems has focused on getting these utilities “right”. One alternative worth exploring is the idea of using endogenous fitness; that is, reinforcing successful performance with “resources” that rules need in order to reproduce. Under this regime, the best rules are those that accumulate the most resources over their lifetime and, consequently, have the most offspring. This paper describes a classifier system designed along these lines. Rules have no associated utility measure. Instead, each rule has one or more reservoirs that can be used to store resources. When enough resources have been accumulated, a rule utilizes some of its resources to reproduce and the reservoir level is reduced accordingly. Preliminary tests of this system on the multiplexor problem show that it performs as well as utility-based classifier systems such as XCS.

1 Introduction

One of the most attractive aspects of the classifier system framework is the way it treats rules as tentative hypotheses subject to testing and confirmation via experience with the environment [4]. A key requirement underlying this capability is a computational approach to assessing the strength of each hypothesis. In classifier systems, the strength of each hypothesis is determined from experienced-based predictions of rule performance. Hypothesis strength is used to provide a sound basis for assessing the fitness of rules, making rules a source of building blocks for generating plausible new hypotheses. Classifier system implementations have traditionally put this idea into practice by calculating parameters — such as predicted payoff, accuracy, payoff-derived strength, etc. — that explicitly estimate the utility and fitness of rules. We are investigating a classifier system that determines the utility and fitness of rules endogenously, without computing explicit estimates.

This research was prompted in part by difficulties encountered over the years trying to devise quantitative performance measures and fitness functions for

traditional classifier systems. In a complex and uncertain environment, a single measure of what makes a hypothesis useful or reliable can be hard to come by. For some problem domains, it is sufficient if a system finds hypotheses that minimize the total number of classification errors. Sometimes, though, it is important to scrutinize hypotheses more carefully, insisting on properties such as a low rate of false positive errors. In other cases, the challenge could be to find clusters that cover the input space subject to various constraints. Designing a single performance measure that accommodates such a wide variety of performance requirements, including their context dependent interactions, can be a significant engineering challenge.

Holland recognized that many of the adaptive interactions that arise in classifier systems can be studied more directly in a simpler framework. The Echo class of models [5] provides such a setting. Echo offers a framework in which simply-defined agents interact in carefully structured ways that are more concrete than the general-purpose, pattern-directed procedures used in classifier systems. Another interesting difference between Echo and classifier systems is that there are no explicit fitness functions for the agents in Echo. The fitness of agents in Echo is determined endogenously. Agents interact with each other and with the environment to acquire the resources needed to survive and reproduce. While fitness is a function of the agent and the overall context the agent finds itself in, that function never has to be explicitly defined as a quantitative measure of adaptive proficiency.

We hypothesize that endogenous fitness can provide a similar advantage in the classifier system framework. This paper introduces ideas about how to implement endogenous fitness in classifier systems, and gives preliminary results on how well those ideas work. We begin with a brief discussion of why we expect endogenous fitness to be useful in classifier systems. That discussion is followed by a description of our initial implementation, and results from experiments with the Boolean multiplexor problem.

2 Endogenous Fitness In Classifier Systems

Our focus in this paper is on using endogenous fitness to address the issue of generalization accuracy. As Wilson [8] has pointed out, traditional classifier systems have no way of distinguishing between an accurate classifier having some modest payoff value, and an inaccurate classifier whose average payoff is equal to that value. Moreover, traditional classifier systems have few explicit mechanisms in place to promote the discovery and proliferation of accurate rules. As a result, overly general rules continually threaten to take over the population, since they match more messages than their less frequently activated, but potentially more accurate, counterparts.

Though measuring the accuracy of a rule can itself be a tricky endeavor [6], there are ways to define rule utilities so that overly general rules are less fit and less likely to dominate the population. For example, one of Holland's earliest discussions of classifier systems emphasized the importance of using prediction

errors to identify rules that overestimate their expected payoff [3]. In addition, Booker [2] defined a consistency parameter, based on the mean-squared prediction error, that was used to attenuate rule fitness and provide an advantage for accurate rules. More recently, Wilson [8] has devised the XCS system that uses a fitness criterion based solely on minimizing prediction error. The success of XCS demonstrates how important it is to effectively address the accuracy issue in any successful classifier system implementation.

The rationale for using endogenous fitness to address the problem of generalization accuracy is based on the following intuition: If rules need resources to reproduce, and accurate rules accumulate resources and reproduce at a faster rate than inaccurate rules, then accurate rules will tend to take over the population. The key is to find simple ways to adjust the relative rate at which accurate and inaccurate rules accumulate the resources needed to reproduce. Our approach therefore focuses on how to design such a competition for resources between accurate and inaccurate rules.

To begin with a simple case, assume that we are dealing with a stimulus-response problem in which the rules advocating an action on a given time step (i.e., the action set) take all of the credit for whatever reinforcement (“good” or “bad”) is received. Assume further that each reinforcement event leads to the distribution of some nominal resources among those rules, and that the acquired resources accumulate over time in internal reservoirs. Following the way resources are used in Echo [5], we specify that a rule needs to accumulate resources in excess of some threshold level before it is allowed to reproduce. Note that the expected net accumulation of resources, reflecting the net balance of “good” and “bad” reinforcement received, does not depend on how frequently a rule is active. It depends most directly on the probability of reinforcement and, therefore, on the probability that the rule is “correct”. Consequently, there is some hope that we can sidestep the issue faced by systems employing explicit fitness functions which must try to compensate for the fact that general rules typically have more opportunities to reproduce than specialized rules.

While this intuitive picture of how to proceed is fairly straightforward, there are many pragmatic details that need to be sorted out. The most important details are those governing the rate of accumulation of resources and the frequency of reproduction. Mathematically speaking, this approach tries to exploit the expected fluctuations in reservoir levels given a sequence of reinforcement events correlated with a rule’s responses. The sequence of changes in reservoir levels can be viewed as a generalized random walk. Given that there are many analytical results available concerning the behavior of random walks and other fluctuation phenomena, it is reasonable to expect that some version of this endogenous fitness scheme should be amenable to mathematical analysis¹. As a prelude to a formal analysis, we have conducted a series of empirical experiments to help determine an appropriate way to pose the problem in the classifier system setting. The remainder of this paper describes those experiments and the classifier system implementations that appear to be on the right track.

¹ This is not meant to suggest that such an analysis will be easy.

3 System Description

The classifier system used in this study has a fairly conventional design, borrowing elements from previous work on GOFER [1,2] and XCS [8]. The population of classifiers has a fixed size \mathcal{N} . Each classifier in the population has a single condition on the left side and a single action on the right side. In a departure from previous systems, there are no performance estimates associated with classifiers. Instead, each classifier ξ has two associated reservoirs: the $\Delta_+(\xi)$ reservoir that stores resources obtained from “good” reinforcement events, and the $\Delta_-(\xi)$ reservoir that stores resources obtained from “bad” reinforcement events. In all of our experiments the reservoirs were initialized to be empty and the initial classifiers were generated at random. The only other parameter stored with each classifier is its age $\alpha(\xi)$, which is used in the procedure for deleting classifiers.

3.1 Performance System

The system performance cycle is fairly routine. For each input message i , the system first determines the set of classifiers \mathbf{M} eligible to classify the message. Matching classifiers are always included in \mathbf{M} . Following the procedures in GOFER, if there are fewer than \mathcal{N}_m matching classifiers available, classifiers with the highest partial match scores are deterministically selected to fill out \mathbf{M} . We use the simple partial match score

$$\mu(\xi, i) = \begin{cases} s + l & \text{if } \xi \text{ matches the message } i \\ l - n & \text{otherwise} \end{cases}$$

where l is the length of the input condition in ξ , s is the specificity², and n is the number of positions where the condition doesn’t match the message.

For each action a represented in \mathbf{M} , the system computes an *action mandate* that is similar in intent to the system prediction computed in XCS. The action mandate is supposed to capture the system’s knowledge about the likelihood of a “good” outcome if action a is chosen. There are several ways this could be computed. We currently use the relative amount of resources in the $\Delta_+(\xi)$ and $\Delta_-(\xi)$ reservoirs as an indicator of expected outcome. Each classifier ξ in \mathbf{M} contributes

$$\lambda(\xi) = \frac{|\Delta_+(\xi)|}{(|\Delta_+(\xi)| + |\Delta_-(\xi)|)}$$

to the mandate for its action. The rationale for this particular approach is to give more weight to those actions that, based on previous experience, have the highest likelihood of being followed by a “good” reinforcement event. This contribution from each rule is added to an *action selection array* and, as in XCS, an action is selected using one of many possible selection methods. The members of \mathbf{M} that agree with the selected action constitute the *action set* \mathbf{A} . The system then sends that action to the effectors, and the environment may respond with reinforcement.

² The number of non-#’s in the input condition

3.2 Reinforcement

For convenience, we assume for now that a zero reward indicates no reinforcement event has occurred. When a non-zero reward is received, a fixed amount of resource $\mathbf{R} \geq 0$ is made available to the classifiers in the action set. As in the GOFER system, each action set classifier is eligible to receive a share of that resource. We have experimented with several ways of allocating the resource among the classifiers in \mathbf{A} . The most effective method so far divides the resource based on the largest likelihood that each classifier assigns to one of the outcomes. This likelihood is estimated by

$$\lambda_*(\xi) = \frac{|\Delta_*(\xi)|}{(|\Delta_+(\xi)| + |\Delta_-(\xi)|)}$$

where $\Delta_*(\xi)$ is the reservoir containing the largest amount of resources. Each classifier in \mathbf{A} receives a share of the resource given by

$$\rho(\xi) = \left(\frac{\lambda_*(\xi)}{\Lambda} \right) \mathbf{R}$$

where Λ is the total likelihood in \mathbf{A} . When the reinforcement event is “good”, $\rho(\xi)$ is added to $\Delta_+(\xi)$; otherwise, it is added to $\Delta_-(\xi)$. The idea is to bias the allocation of resources in favor of those classifiers that provide the most decisive hypothesis about the outcome associated with the selected action.

Under this regime, rules that are consistently associated with only one type of outcome will quickly achieve a large net accumulation of resources in the $\Delta_*(\xi)$ reservoir since all of their resources are stored in one place. Conversely, rules associated with both outcomes will distribute their resources over both reservoirs, taking longer to attain a large net accumulation in $\Delta_*(\xi)$. This is significant because the frequency of reproduction is tied to the net accumulation of resources in the $\Delta_*(\xi)$ reservoir.

3.3 Rule Discovery

After the rule reservoirs have been updated, any classifier in \mathbf{M} having a sufficient excess of resources in the $\Delta_*(\xi)$ reservoir becomes eligible to reproduce. An excess of resources is indicated by

$$|\Delta_+(\xi) - \Delta_-(\xi)| > \tau$$

for some threshold τ . When reinforcement is correlated with “correct” and “incorrect” actions, the frequency of satisfying this reproduction criterion is correlated with the frequency that a rule’s response is correct. Rules that are consistently correct (or consistently incorrect) will consequently enjoy a reproductive advantage over rules that are inconsistent. Note that this notion of directly assessing the performance of the decision policy leads to a policy-level view of what it means for a rule to be accurate. In GOFER, XCS and other systems that

try to estimate the expected reward associated with a state or state-action pair, accuracy refers to the precision of those value estimates.

If there is more than one classifier in \mathbf{M} eligible to reproduce on a given cycle, all eligible classifiers are designated as parents and allowed to produce one copy of themselves. Parents then have their $\Delta_*(\xi)$ reservoir decremented by τ , which can be viewed as the cost of generating an offspring. The reproduced copies are modified by mutation and crossover, and the resulting offspring are inserted into the population. Classifiers are stochastically selected for deletion based on their age, so that older classifiers are more likely to be chosen. Note that this scheme triggers rule discovery without using any extra bookkeeping or monitoring of the state of the system.

4 Empirical Results

As a simple initial test of how well this system works, we apply it to the familiar Boolean multiplexor problem. The value of the multiplexor function is computed by using the first n bits of an l -bit string as an index. That index selects one of the remaining 2^n bits in the string (where $n + 2^n = l$), and the selected bit is returned as the function value. We initially restrict our attention to the case $n = 2$ (the 6-bit multiplexor). For this problem, the following set of rules provides a complete solution:

```

000### ==> 0   10##0# ==> 0
001### ==> 1   10##1# ==> 1
01#0## ==> 0   11###0 ==> 0
01#1## ==> 1   11###1 ==> 1

```

Each of these rules is maximally general in the sense that no more #’s can be added to their input conditions without introducing performance errors.

In all experiments, unless otherwise noted, the system constants were as follows: $\mathcal{N} = 400$, $\mathcal{N}_m = 16$, $\mathbf{R} = 500$, $\tau = 1000$, initial reservoir levels of 0 for new offspring, a mutation rate of 0.01, and a crossover rate of 1.0. We identify a correct system response as a “good” event and an incorrect response as a “bad” event. As a convenient simplification, we initially use a positive reward (+1000) to signal a “good” event and a negative reward (−1000) to signal a “bad” event. The initial version of the endogenous fitness algorithm only checks the sign of the reward, though, so the magnitude of these rewards is ignored. In subsequent sections we will describe a generalization of this basic scheme in which rewards with different magnitudes can be used and the sign of the reward does not automatically determine if the response is “good” or “bad”.

Performance was measured by the proportion of correct decisions over a learning epoch consisting of 50 randomly generated input strings. Each experiment was run for 200 epochs (10,000 input strings). In order to facilitate comparisons with XCS on this problem, we used Wilson’s [8] action-selection regime. This regime makes a random (probability 0.5) choice between “exploit” mode — where the system selects the best action as indicated by the action mandate

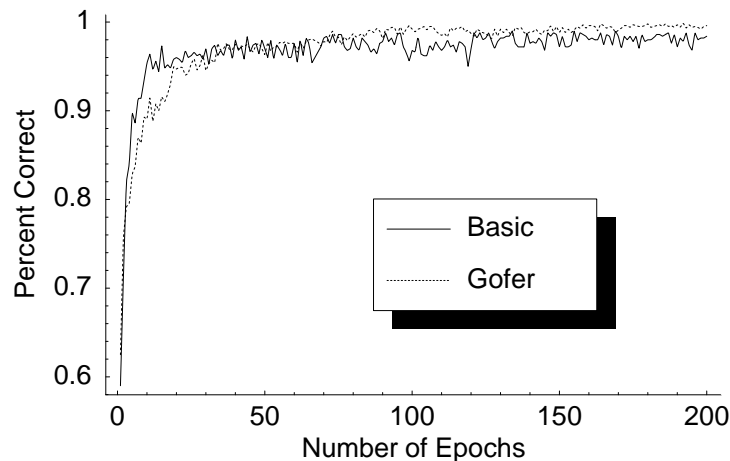


Fig. 1. Initial performance on 6-bit multiplexor

array — and “explore” mode where the system selects an action at random³. No rule discovery operations occur in exploit mode. All results are averaged over 10 independent runs.

4.1 Initial Test Of The Basic System

The results of applying the system described above to the 6-bit multiplexor problem are shown in Figure 1.

This initial implementation of the endogenous fitness algorithm has a performance trajectory that is very similar to the performance curve of the more conventional GOFER approach. At the end of 200 epochs, performance was approximately 98.2% correct as compared to 99.6% for GOFER. Direct comparison with the GOFER performance curve is somewhat misleading, since GOFER used a roulette-wheel action selection scheme that allowed learning on every trial. Moreover, GOFER only tried to learn the 8 maximally general classifiers that are always correct. Here, in order to assess generalization accuracy, we make our system learn all 16 maximally general classifiers, both those that are 100% correct and those that are 100% incorrect. Nevertheless, the performance of the endogenous fitness approach appears to be reasonably good on this task.

When we examine the rules that the endogenous fitness algorithm has learned after 200 epochs, however, we see that there is considerable room for improvement. Figure 2 shows the most prevalent rules in a population of classifiers at the end of a typical run. The numbers in parentheses after each rule give the number of rule instances and the reward expected when that rule controls the system

³ During explore trials we record a correct decision whenever the system would have made the correct “exploit” decision

```

(State = ##1#1#) ==> 0 (8,-407.1)
(State = ##11##) ==> 0 (9,-89.9)
(State = #0##1#) ==> 0 (9,44.6)
(State = #1#1#1) ==> 1 (11,1000.0)
(State = 000###) ==> 1 (26,-1000.0)
(State = 01#1##) ==> 0 (14,-1000.0)
(State = 01#1##) ==> 1 (14,1000.0)
(State = 1###0#) ==> 1 (9,-95.4)

```

Fig. 2. Typical set of rules learned by the initial endogenous fitness scheme

response⁴. The system typically learns all of the maximally general classifiers, but only a few manage to attain a sizable foothold in the population during the first 200 epochs. In retrospect this is not too surprising. Over-general rules that help the system perform better than random in early epochs do not suffer an overwhelming disadvantage in the competition for resources with maximally general classifiers. The competition for resources is governed by the likelihoods $\lambda_*(\xi)$. An over-general classifier like

```
1###0# ==> 1
```

is correct on only 4 of the 16 messages it matches, and so we might naively⁵ expect it to have an associated likelihood of $\lambda_*(\xi) = 0.75$. While this is smaller than the corresponding expectation of $\lambda_*(\xi) = 1.0$ for a maximally general classifier, it takes time for this difference to lead to the demise of the over-general classifier, especially when the over-general classifier has a significant number of instances in the population. Eventually the difference in likelihoods does take its toll. When we extend the experiments to 400 epochs we see that by 350 epochs the maximally general classifiers have taken over the population and the performance curves are steady at 100% correct. However, this is about twice as long as it takes systems like GOFER and XCS to generate a comparable population of maximally general rules.

4.2 Generating A Better Covering

One of the reasons that the basic system fails to quickly find and exploit all of the maximal generalizations is that the selection pressure against overly general or overly specific classifiers is weaker here than in GOFER and XCS. The most efficient population of classifiers for solving the multiplexor problem is one that consists solely of the 16 maximally general classifiers. These classifiers *cover*

⁴ The statistics about expected reward were computed for diagnostic purposes only. They were not used by the performance system.

⁵ Even if the algorithm were formulated as a simple, symmetric random walk, chance fluctuations make it unlikely that we would observe such “intuitively obvious” reservoir levels for any given classifier.

the set of input strings in the sense that they correctly indicate the outcome associated with each action and each input. The maximally general classifiers provide an *efficient* covering in the sense that they are the most compact set of rules that covers all of the input strings. The endogenous fitness scheme, in its current form, is not explicitly designed to learn an efficient covering. We can improve the performance of the algorithm by finding a way to provide more direct guidance about how to generate such a covering⁶.

Smith *et al* [7] describe an intriguing approach to generating selection pressures for efficient coverings. Their technique is based on a simple model of the immune system in which antibodies compete with each other to match antigens. Both antigens and antibodies are represented as binary strings. The fitness of each antibody is determined by the number of times it is selected as the winner of a competition. Experiments show that, under this fitness regime, a genetic algorithm (GA) can generate and maintain a diverse population of antibodies that covers the set of antigens. Moreover, each antibody has a frequency in the population proportional to the frequency with which the corresponding antigens it covers are sampled. An analysis of this mechanism shows that it is strongly related to the familiar fitness-sharing schemes often used in GA implementations to solve multimodal optimization problems.

We can exploit this approach for our purposes by using the probability of winning such a competition to help bias the distribution of resources in \mathbf{A} . More specifically, assume that we want to model the results of the following competition for each input message:

1. Select a random sample of size r without replacement from \mathbf{M} .
2. Choose the classifier ξ in \mathbf{A} with the largest value of $\lambda_*(\xi)$ in the sample as the winner on this iteration. Ties are broken at random. If there are no classifiers from \mathbf{A} in the sample, then there is no winner and we repeat step 1.
3. Add $\lambda_*(\xi)$ to the total competition score for ξ .
4. Repeat the process for several iterations. The classifier that accumulates the highest overall score is the winner of the competition.

Smith *et al* show that the probability of winning this kind of competition can be expressed in terms of hypergeometric probabilities. The hypergeometric distribution $h(k, r, n, m)$, given by

$$h(k, r, n, m) = \frac{\binom{m}{k} \binom{n-m}{r-k}}{\binom{n}{r}}$$

is used to compute the probability that exactly k red balls will be in a sample of size r , drawn without replacement from a population of size n that contains m

⁶ Since we rely on a genetic algorithm for rule discovery, we avoid doing anything that over-zealously eliminates diversity from the population. Diversity is the “grist for the mill” in genetic search.

red balls. In order for a classifier ξ to win the competition for a given message, no classifier φ with $\lambda_*(\varphi) > \lambda_*(\xi)$ can occur in the sample. Moreover, whenever there are classifiers φ with $\lambda_*(\varphi) = \lambda_*(\xi)$ in the sample, ξ has to be the one chosen by the tie-breaking procedure. The probability that these events all occur is given by

$$\mathbf{H}(\xi) = \frac{h(0, r, |\mathbf{M}|, G(\xi))(1 - h(0, r, |\mathbf{M}| - G(\xi), F(\xi)))}{F(\xi)}$$

where $F(\xi)$ is the number of classifiers φ in \mathbf{A} with $\lambda_*(\varphi) = \lambda_*(\xi)$, and $G(\xi)$ is the number classifiers φ in \mathbf{A} with $\lambda_*(\varphi) > \lambda_*(\xi)$. Given the way sample size controls the extent of resource sharing in this algorithm [7], the value $r = 0.5|\mathbf{M}|$ appears to be a reasonable choice for our classifier system implementation.

Directly computing the hypergeometric probabilities is a cost-effective alternative to explicitly simulating the multi-stage competition on each classifier system cycle. The log gamma function, available as a subroutine in standard math libraries, provides a direct way to compute factorials and binomial coefficients based on the fact that

$$\Gamma_{\ln}(n) \stackrel{\text{def}}{=} \log(\Gamma(n)) = \log((n-1)!)$$

for integers n . The hypergeometric probabilities $h(0, r, n, m)$ that we need are given by

$$h(0, r, n, m) = \exp(\Gamma_{\ln}(n - m + 1) + \Gamma_{\ln}(n - r + 1) - \Gamma_{\ln}(n - m - r + 1) - \Gamma_{\ln}(n + 1))$$

where \exp is the exponential function⁷. We use these probabilities to exert pressure for a more efficient covering by modifying the endogenous fitness scheme to use

$$\tilde{\rho}(\xi) = \left(\frac{\lambda_*(\xi)\mathbf{H}(\xi)}{\sum_{\xi \in \mathbf{A}} \lambda_*(\xi)\mathbf{H}(\xi)} \right) \mathbf{R}$$

when allocating resources to \mathbf{A} . This biases the allocation of resources in the desired way, making elements of a good covering more likely to reproduce.

This modified version of the endogenous fitness scheme was tested on the 6-bit multiplexor problem. The same parameters were used as before, except that we used an increased value $\mathbf{R} = 5,000$ to compensate for the change of scale caused by using the hypergeometric probabilities in the computation of $\tilde{\rho}$. Figure 3 shows that the selection pressure for a good covering does indeed improve performance.

Figure 4 shows that the performance improvements include a much better capability to generate an efficient covering. The system reliably finds all of the

⁷ This computation is made more efficient by noting that we will use at most \mathcal{N} different values of the factorial function. Accordingly, values can be computed once using the log gamma function and stored in a table for retrieval on subsequent function calls.

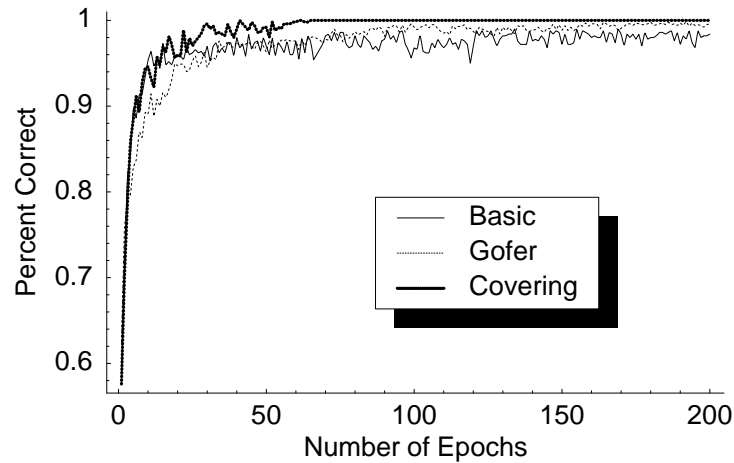


Fig. 3. Performance of the improved covering algorithm on the 6-bit multiplexor

```

(State = 0#11##) ==> 1 (8,1000.0)
(State = 001###) ==> 0 (17,-1000.0)
(State = 001###) ==> 1 (23,1000.0)
(State = 000###) ==> 0 (21,1000.0)
(State = 000###) ==> 1 (16,-1000.0)
(State = 01#0##) ==> 0 (28,1000.0)
(State = 01#0##) ==> 1 (19,-1000.0)
(State = 01#1##) ==> 0 (9,-1000.0)
(State = 01#1##) ==> 1 (12,1000.0)
(State = 10##1#) ==> 0 (24,-1000.0)
(State = 10##1#) ==> 1 (16,1000.0)
(State = 10##0#) ==> 0 (25,1000.0)
(State = 10##0#) ==> 1 (19,-1000.0)
(State = 11###1) ==> 0 (28,-1000.0)
(State = 11###1) ==> 1 (17,1000.0)
(State = 11###0) ==> 0 (20,1000.0)
(State = 11###0) ==> 1 (24,-1000.0)

```

Fig. 4. Typical set of rules learned by the endogenous fitness scheme modified for improved covering

maximally general classifiers, and they rapidly establish a strong presence in the population. Populations of this kind are comparable to what we expect from systems like XCS.

4.3 Arbitrary Reinforcement Schemes

Up to this point, we have focused on developing the endogenous fitness mechanisms in a simplified setting. The system has only been required to learn to distinguish between correct and incorrect responses given training experiences that provide explicit information about response categories. Most reinforcement learning problems, though, impose requirements that are much different. Generally speaking, a reinforcement learning problem forces a learning system to use scalar rewards without any accompanying labels or supplemental information as input. To put it simply, the system must learn to choose the action that results in the most reward in a given situation. Any approach to solving reinforcement learning problems that cannot meet this challenge is severely limited.

There is a fairly straightforward way to generalize the endogenous fitness algorithm we have described to handle such arbitrary reinforcement schemes. The key is to recognize that the system does not really need external guidance about which reservoir, $\Delta_+(\xi)$ or $\Delta_-(\xi)$, should be used to store a resource. Instead of thinking in absolute terms about “good” or “bad” outcomes, we can think in relative terms about “better” or “worse” outcomes given some reference level for rewards. The average reward is an easily computed reference level we can use for this purpose. When a reinforcement event involves above average reward, the resource can be added to the $\Delta_+(\xi)$ reservoir; otherwise, it goes to the $\Delta_-(\xi)$ reservoir.

This idea suggests that we make the following changes to our implementation of the endogenous fitness algorithm:

- Each classifier ξ gets a new parameter $\pi(\xi)$ that estimates the average reward available when ξ is included in \mathbf{M} . We have experimented with several approaches to computing $\pi(\xi)$. The most effective method so far computes it as the sample average of the rewards received on “explore” trials. This computation uses another new parameter $\nu(\xi)$ that counts the number of times ξ has been included in \mathbf{M} on an “explore” trial. The $\pi(\xi)$ parameter is revised using the simple update rule

$$\pi_t(\xi) = \begin{cases} \frac{(\pi_{t-1}(\xi)\nu_{t-1}(\xi)) + \mathcal{R}_t}{\nu_t(\xi)} & \text{if } \nu_t(\xi) \neq \nu_{t-1}(\xi) \\ \pi_{t-1}(\xi) & \text{otherwise} \end{cases}$$

where \mathcal{R}_t is the reward at time t , $\pi_0(\xi) = 0$, and $\nu_0(\xi) = 0$. In terms of standard approaches to solving reinforcement learning problems, we can think of $\pi(\xi)$ as an estimate for the utility of the states that ξ is used in. Note that it is not an estimate for the utility of ξ .

- The mean value of these estimates, weighted by likelihood, is used to determine the reference level for interpreting reinforcement events. More specifically, we use the members of \mathbf{M} to collectively estimate the utility of the

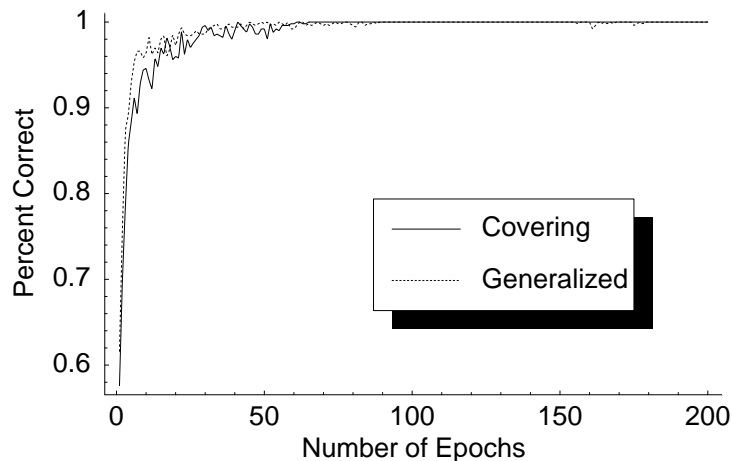


Fig. 5. Performance using the (1000,-1000) reward scheme

current state as

$$\Pi = \frac{\sum_{\xi \in \mathbf{M}} \lambda_*(\xi) \pi(\xi)}{\sum_{\xi \in \mathbf{M}} \lambda_*(\xi)}$$

and determine if the current reward is above or below average by comparing it to Π . When the reward is above average, the resource is added to $\Delta_+(\xi)$. When the reward is below average, it is added to $\Delta_-(\xi)$.

- We linearly scale the resource \mathbf{R} available on each time step to reflect how much the reward deviates from average, using

$$\mathbf{R} \left(0.5 + \frac{|\mathcal{R} - \Pi|}{2\delta} \right)$$

where δ is the range of the scalar rewards and represents the maximum possible deviation (δ is a system parameter that must be provided). This scaling yields $\mathbf{R}/2$ for average events and \mathbf{R} for events involving a deviation of size δ . Given two classifiers that are consistently associated with above (or below) average rewards, this modification to \mathbf{R} provides a modest selective advantage to the classifier that is best (or worst).

All other aspects of the endogenous fitness algorithm remain the same, including the aforementioned modifications for improved covering.

We conducted several experiments to determine if this generalized version of the endogenous fitness algorithm can learn the multiplexor problem given arbitrary reinforcement schemes that do not indicate correct or incorrect responses. Initially, we used two simple reinforcement schemes: a (1000,-1000) reward scheme giving 1000 for right answers and -1000 for wrong answers; and, a (1000,500) reward scheme. As shown in Figures 5 and 6, the performance of the

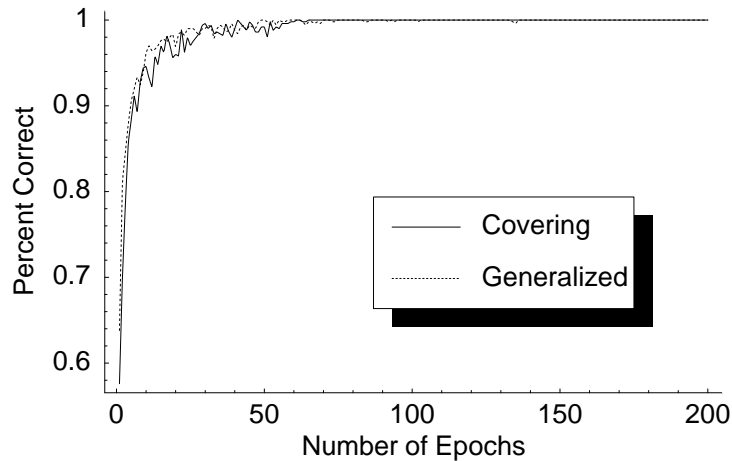


Fig. 6. Performance using the (1000,500) reward scheme

generalized endogenous fitness algorithm using these reward schemes is almost identical the previous performance achieved using the covering enhancements with explicit designations of correct and incorrect actions.

A more challenging test is offered by Wilson’s [8] layered payoff landscape for the multiplexor problem. This payoff landscape provides different rewards for each set of eight strings that matches one of the maximally general classifiers. The reward scheme for right and wrong answers associated with each set of strings is summarized below:

| | |
|------------------------|-------------------------|
| 000### ==> 0 (300,0) | 10##0# ==> 0 (700,400) |
| 001### ==> 1 (400,100) | 10##1# ==> 1 (800,500) |
| 01#0## ==> 0 (500,200) | 11###0 ==> 0 (900,600) |
| 01#1## ==> 1 (600,300) | 11###1 ==> 1 (1000,700) |

Figure 7 shows that the generalized endogenous fitness algorithm does learn to solve the problem using this payoff landscape, though at a slightly slower pace than the previous version. One factor contributing to this discrepancy is the way we scaled resources. In this payoff landscape there is a different average payoff for each set of eight strings. The payoff range in each set is 300. However, since we used the overall payoff range of 1000 as δ when we scaled resources, reservoir levels were increased in increments less than \mathbf{R} on each time step. This means it takes longer for reservoir levels to exceed their threshold for reproduction and, consequently, longer for the system to learn how to solve the problem. An additional complication is introduced by the presence of overgeneral rules, which can distort estimates of the state utility H . These effects all need to be examined more carefully, and are an important topic for further research.

As a final test, we looked at the performance of our system on the more difficult 11-bit multiplexor problem. The only change we made to the system is

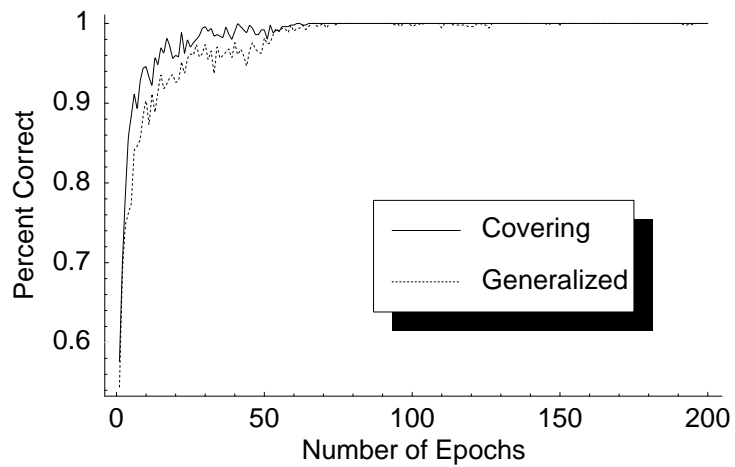


Fig. 7. Performance using Wilson's layered payoff landscape

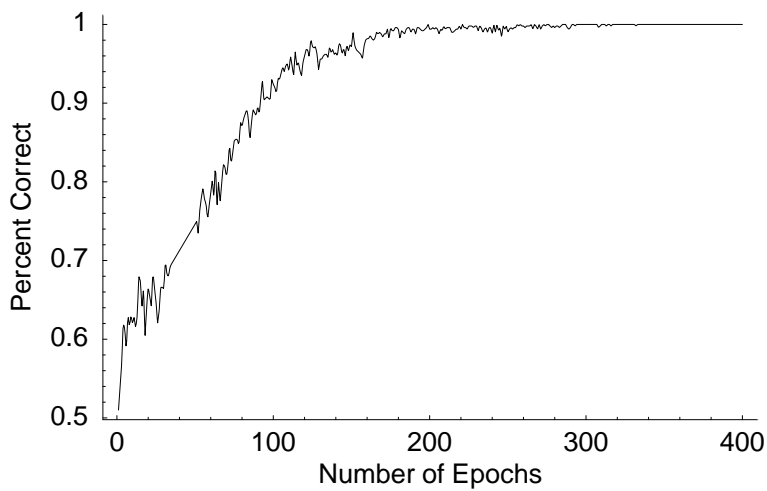


Fig. 8. Performance on the 11-bit multiplexor

```

(State = 0000#####) ==> 0 (28,1000.0)
(State = 0000#####) ==> 1 (13,500.0)
(State = 0001#####) ==> 0 (19,500.0)
(State = 0001#####) ==> 1 (47,1000.0)
(State = 001#0#####) ==> 0 (44,1000.0)
(State = 001#0#####) ==> 1 (9,500.0)
(State = 001#1#####) ==> 0 (14,500.0)
(State = 001#1#####) ==> 1 (35,1000.0)
(State = 010##0#####) ==> 0 (22,1000.0)
(State = 010##0#####) ==> 1 (13,500.0)
(State = 010##1#####) ==> 0 (13,500.0)
(State = 010##1#####) ==> 1 (32,1000.0)
(State = 011###1#####) ==> 0 (15,500.0)
(State = 011###1#####) ==> 1 (23,1000.0)
(State = 011###0####) ==> 0 (30,1000.0)
(State = 011###0####) ==> 1 (10,500.0)
(State = 100####0###) ==> 0 (34,1000.0)
(State = 100####1###) ==> 1 (18,1000.0)
(State = 101#####0##) ==> 0 (29,1000.0)
(State = 101#####0##) ==> 1 (13,500.0)
(State = 101#####1##) ==> 1 (22,1000.0)
(State = 110#####1#) ==> 0 (12,500.0)
(State = 110#####1#) ==> 1 (30,1000.0)
(State = 110#####0#) ==> 0 (33,1000.0)
(State = 110#####0#) ==> 1 (12,500.0)
(State = 111#####1) ==> 0 (9,500.0)
(State = 111#####1) ==> 1 (35,1000.0)
(State = 111#####0) ==> 0 (43,1000.0)
(State = 111#####0) ==> 1 (13,500.0)

```

Fig. 9. Typical set of rules learned for the 11-bit multiplexor

to use a larger population size $\mathcal{N} = 800$. Figure 8 shows the performance using the (1000,500) reward scheme. The system solves the problem after about 12,000 input strings, which is consistent with results reported for XCS [8]. Figure 9 shows that the system generates a population full of maximally general classifiers as we would expect.

5 Summary

The results presented here are modest and preliminary. However, they strongly suggest that endogenous fitness schemes like those described in this paper are worth further investigation. Many important research issues remain to be investigated, particularly those related to multi-step reinforcement learning problems involving delayed rewards. While more experiments are needed to build our intuitions about how this approach can be used, the most important thing to do

is formally analyze how the fluctuations in reservoir levels are translated into reproduction rates. That kind of understanding is vital if these schemes are ever to become reliable approaches for building classifier systems. Our current research efforts are focused on conducting that analysis.

† Acknowledgments This research was funded by the MITRE Mission-Oriented Investigation and Experimentation (MOIE) research program. That support is gratefully acknowledged.

References

1. Booker, L. B. Classifier systems that learn internal world models. *Machine Learning 3* (1988), 161–192.
2. Booker, L. B. Triggered rule discovery in classifier systems. In *Proceedings of the Third International Conference on Genetic Algorithms (ICGA89)* (Fairfax, VA, 1989), J. D. Schaffer, Ed., Morgan Kaufmann, pp. 265–274.
3. Holland, J. H. Adaptation. In *Progress in theoretical biology*, R. Rosen and F. M. Snell, Eds., vol. 4. Academic Press, New York, 1976.
4. Holland, J. H. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In *Machine learning: An artificial intelligence approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds., vol. II. Morgan Kaufmann, Los Altos, CA, 1986, ch. 20, pp. 593–623.
5. Holland, J. H. Echoing emergence: Objectives, rough definitions, and speculations for Echo-class models. In *Complexity: Metaphors, Models, and Reality*, G. Cowan, D. Pines, and D. Melzner, Eds., vol. XIX of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, Reading, MA, 1994, pp. 309–342.
6. Horn, J., Goldberg, D. E., and Deb, K. Implicit niching in a learning classifier system: Nature’s way. *Evolutionary Computation 2*, 1 (1994), 37–66.
7. Smith, R. E., Forrest, S., and Perelson, A. S. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation 1*, 2 (1993), 127–149.
8. Wilson, S. W. Classifier fitness based on accuracy. *Evolutionary Computation 3*, 2 (1995), 149–175.