ZCS: A Zeroth Level Classifier System

Stewart W. Wilson
The Rowland Institute for Science
100 Edwin H. Land Blvd.
Cambridge, MA 02142
wilson@smith.rowland.org

Abstract

A basic classifier system, ZCS, is presented that keeps much of Holland's original framework but simplifies it to increase understandability and performance. ZCS's relation to Q-learning is brought out, and their performances compared in environments of two difficulty levels. Extensions to ZCS are proposed for temporary memory, better action selection, more efficient use of the genetic algorithm, and more general classifier representation.

Keywords

Classifier systems, Q-learning, temporary memory, action selection, restricted mating, s-classifiers, genetic programming.

1. Introduction

A *classifier system* is a learning system in which a set of condition-action rules called classifiers compete to control the system and gain credit based on the system's receipt of reinforcement from the environment. A classifier's cumulative credit, termed strength, determines its influence in the control competition and in an evolutionary process using a genetic algorithm in which new, plausibly better, classifiers are generated from strong existing ones, and weak classifiers are discarded. The original classifier system concept is due to Holland, who described it most completely in Holland (1986). Responding to perceived shortcomings of existing artificial intelligence systems, particularly with regard to adaptation and practical induction, Holland laid out a competitive/cooperative message-passing framework addressing desiderata including temporal credit assignment under conditions of sparse or delayed reinforcement, distributed and generalizable representations of complex categories, default responses subject to exceptions, and graceful adaptation of system knowledge through gradual confirmation/disconfirmation of hypotheses. Though a number of researchers were inspired by Holland's framework to investigate classifier systems (for a review, see Wilson & Goldberg, 1989) and it had some influence on the related field of reinforcement learning (Barto, 1992), efforts to realize the framework's potential have met with mixed success, primarily due to difficulty understanding the many interactions of the classifier system mechanisms that Holland outlined. The most successful studies tended, in fact, to simplify and reduce the canonical framework, permitting better understanding of the mechanisms that remained.

Recently, we set out to carry this simplification to the point where all remaining processes might be well understood, while retaining what we deemed to be the essence of the classifier system idea: a system capable of action through time to obtain external reinforcement,

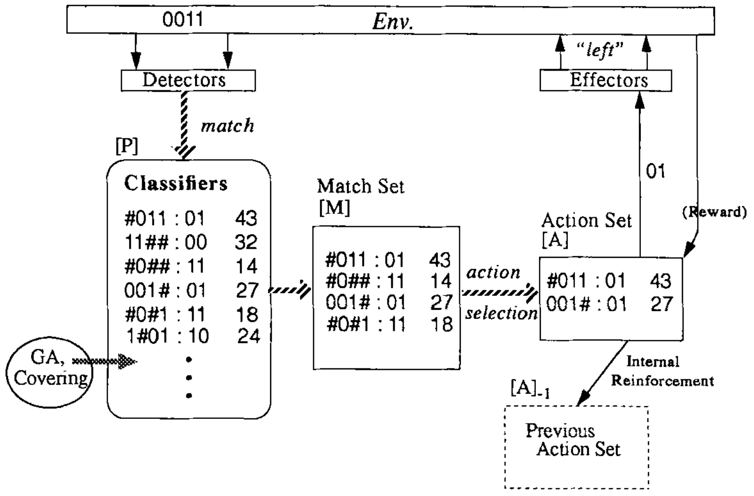

Figure 1. Schematic illustration of ZCS.

based on an evolving set of internally reinforced, generalizable condition-action rules. The result, a sort of "zeroth-level" classifier system, ZCS, has points of analytical contact with systems studied under the heading of reinforcement learning, and appears to provide a viable foundation for building toward the aims of Holland's full framework. The next section of this paper presents a description of ZCS sufficient to allow implementation. Section 3 gives results in two experimental environments. In Section 4, the relationship of ZCS to the technique called Q-learning is discussed. Section 5 suggests extensions to ZCS, including simple temporary memory, more sophisticated action selection, a niche genetic algorithm, and a general representation for classifier conditions. Section 6 concludes by summarizing advantages of ZCS as a reinforcement learner and, with respect to Holland's full framework, notes the price in elements given short shrift.

2. Description of ZCS

Figure 1 gives a broad picture of ZCS. The system is seen in interaction with an environment via detectors for sensory input and effectors for motor actions. In addition, the environment at times provides a scalar reinforcement, here termed reward. The basic idea of a system sensing and learning to act in an environment so as to gain reward appeared early in Holland's work (Holland, 1976). It has more recently become known as the reinforcement learning problem and encompasses many aspects of the general problem of intelligence (Sutton, 1992). ZCS descends from the classifier system developed in Wilson (1985) but differs significantly in omitting several heuristics and operators that aided that system. These included: a look-ahead classifier creation heuristic, a statistic estimating distance to reward kept by each classifier and used in the performance cycle and elsewhere, and a genetic interference operator.

Within Figure 1, the box labeled [P] (population) contains the system's current set of classifiers. They are built on the binary alphabet, plus the "don't care" symbol #; a colon separates the condition and action. The condition, in this example of length four, is a single conjunct to be matched against the indicated detector bits; the action part encodes a motor action. Shown associated with each classifier is a scalar strength. As is apparent, and unlike Holland's framework, ZCS has no internal message list. It is thus incapable of temporary memory and so cannot act on accumulated information detected on any previous time-step, or according to internally generated intentions or controls.

The use of binary detector variables is a restriction in the interest of simplicity, the detectors being regarded either as computing the presence or absence of input features or as thresholding analog input variables—the canonical framework makes the same assumption. While this finesses the problem of learning the features or thresholds in the first place, ZCS still has to learn effective classifier conditions, that is, feature-like combinations of detector output bits. The use of discrete actions is also a simplification, since many creature actions range over a continuum (e.g., turn a little bit left). Classifier systems will ultimately need to deal with both continuous inputs and outputs; research in fuzzy classifier systems (Valenzuela-Rendón, 1991; Parodi & Bonelli, 1993) is one step in that direction.

In the *performance*, or sense-act, cycle of ZCS, the condition of each classifier in [P] is compared with the detector string. If the bit at every non-# position in a classifier's condition matches the corresponding bit in the detector string, the condition is satisfied and the classifier becomes a member of the current *match set* [M]. Next, an action is selected from among those advocated by members of [M]. Many schemes are possible and useful, but ZCS employs perhaps the simplest stochastic method: a roulette wheel with sectors sized according to the strengths of members of [M]. Thus, a particular action a is selected with probability equal to the sum of the strengths of the classifiers in [M] that advocate that action, divided by the total strength of classifiers in [M]. Next, an *action set* [A] is formed, consisting of all members of [M] that advocated a. Finally, a is sent to the effector interface, and the corresponding motor action is carried out in the environment.

ZCS's generalization capability is expressed in the action set [A]. In the first place, individual classifiers of [A] may have #s in their conditions, so they match more than one distinct input. But second, the conditions present are generally different: there may be different numbers of #s, or the specified bits may be in different positions. This diversity reflects ZCS's search for the "best" classifiers in each situation; these are, in general, classifiers that have high relative strength while matching a large number of inputs. There appears to be an inherent pressure in ZCS and similar classifier systems toward such classifiers (Wilson, 1987a).

ZCS's *reinforcement*, or credit assignment, cycle centers around [A] and the action set on the previous time-step, $[A]_{-1}$. The procedure goes as follows. First a fixed fraction β ($0 < \beta \leq 1$) of the strength of each member of [A] is deducted from the member's strength and placed in an (initially empty) common "bucket" B. If $S_{[A]}$ is the total strength of members of [A], the effect is to deduct $\beta S_{[A]}$ from $S_{[A]}$ and place it in the bucket. Second, if the system receives an immediate reward r_{imm} from the environment after taking action a, a quantity $\beta r_{imm}/|A|$ is added to the strength of each classifier in [A] ($|A|$ is the number of classifiers in [A]). The effect is to increase $S_{[A]}$ by βr_{imm}. Third, classifiers in $[A]_{-1}$ (if it is nonempty) have their strengths incremented by $\gamma B/|A_{-1}|$, where γ is a discount factor ($0 < \gamma \leq 1$), B is the total amount put in the bucket in step 1, and $|A_{-1}|$ is the number of classifiers in $[A]_{-1}$. Finally, [A] replaces $[A]_{-1}$ and the bucket is emptied.

To see the overall effect of this process on $S_{[A]}$, it is helpful to define $[A]'$ as the action set that *follows* [A] in time. Then the process can be written as a reassignment

$$S_{[A]} \leftarrow S_{[A]} - \beta S_{[A]} + \beta r_{imm} + \beta\gamma S_{[A]'} \tag{1}$$

This expression can be seen to parallel Holland's bucket brigade, with three main exceptions: classifier specificities are ignored; there is no "bid competition"; and the quantity passed by the "brigade" is reduced by a factor $1-\gamma$ at each step. ZCS ignores specificities on the basis of earlier experiments (Wilson, 1988), indicating that their presence in classifier "payments"—the amounts transferred between classifiers—is undesirable. Other experiments reported in that study found no advantage for the bid competition (in which, in effect, the action is selected from among the classifiers in a high-strength subset of [M]). In contrast, a discount factor γ significantly less than 1.0 appears to be essential in Equation (1) for problems of the kind studied here. We found that if γ is omitted or set near 1.0, then dithering, lack of urgency in attaining rewards, and looping behavior occur, which smaller values of the discount factor greatly reduce.

The sharing of the reward and the bucket amount in the second and third reinforcement steps prevents the genetic algorithm from allocating excessive numbers of classifiers to any given rewarding "niche"; for an analysis see the appendix to Wilson (1987a). Note also, in contrast to the canonical framework, that the "payments" in ZCS's bucket brigade do not pass via the intermediary of posted messages, since there is no message list. Instead, the payments go to the previously active classifiers as though each posted the same message, and that message was matched by each of the presently active classifiers. This technique, first used in Wilson (1985), was termed in Goldberg (1989) an *implicit bucket brigade.*

The reinforcement cycle has one further step in which the strengths of classifiers in the *set difference* [M] – [A] are reduced by a small fraction $\tau \cong \beta$. That is, matching classifiers that advocate actions *other* than the selected action are weakened. This causes the system, over many cycles, to become increasingly definite about its action choices in the situations it encounters. In effect, the use of the "tax" τ, combined with roulette wheel action selection, represents an exploration strategy in which selected actions become increasingly likely to be selected, so that exploitation of the apparent best increasingly displaces exploration. However, there are better techniques for approaching the explore/exploit trade-off, one of which will be proposed in Section 5.2. The system's exploration tendency is more properly handled in the performance, not the reinforcement, cycle, and the τ technique should be regarded as provisional.

The *discovery*, or classifier generating, component of ZCS consists of a basic panmictic genetic algorithm supplemented by a covering operation. The GA operates in the background, at an average rate keyed to the performance cycle. Each time it is invoked, the GA selects two classifiers based on strength, copies them to form offspring, crosses and/or mutates the offspring with fixed probabilities, then inserts the offspring into the population. To maintain [P] at a constant size, two classifiers are deleted with probability proportional to the inverse of their strengths. The initial strengths of offspring classifiers are set so as to conserve the total strength of parents and offspring. Half of each parent's strength is deducted from the parent and assigned to its copy. If crossover occurs, the copy strengths are reset to their mean. The rate of GA invocation is problem-dependent. On the average, classifiers should be evaluated (be in some [A]) a sufficient number of times before being selected, in order that their strengths settle to values that approximate steady-state values. If the GA is run too fast, strengths will on average be too noisy; if run too slowly, the system will not achieve its best rate of improvement. In ZCS, the GA rate is controlled by a probability of invocation per time-step, which must be chosen by the user. Techniques for automatic rate control are known (Booker, 1982), based on each classifier's keeping a count of its evaluations.

The covering operation deals with the situation occurring when [M] is empty (i.e., no classifier in [P] matches the input) or when the total strength in [M], $S_{[M]}$, is less than a fraction ϕ of the population mean strength. Covering creates a new classifier whose condition matches the input and contains a probabilistically determined number of #s. The classifier's action is chosen randomly, and the strength is set to the population average. The new classifier is inserted into [P], and a classifier is deleted as in the GA. Then the system forms a new [M] and proceeds as usual. Covering is a relatively crude operation resembling rote learning or imprinting; it does not build on knowledge already in the population, as does the GA. However, the empty [M] condition is inevitable in most systems, and something must be done when it occurs. Rather than merely acting randomly, covering allows the system to act randomly but test a hypothesis (the condition-action relation expressed by the created classifier) at the same time.

The foregoing description of ZCS has mentioned most of the system's parameters. They are summarized below.

N	Population size.
$P_{\#}$	Probability of a # at an allele position in the condition of a classifier created through covering, and in the conditions of classifiers in the initial randomly generated population.
S_0	Strength assigned to each classifier in the initial population.
β	Learning rate for strength updates under the bucket brigade.
γ	Discount factor for the bucket brigade.
τ	Fraction of strength deducted from classifiers in [M] − [A].
χ	Probability of crossover per invocation of the GA.
μ	Probability of mutation per allele in an offspring. Mutation takes 0, 1, # equiprobably into one of the other allowed alleles.
ρ	Average number of new classifiers generated by the GA per time-step of the performance cycle.
ϕ	If the total strength of [M] is less than ϕ times the mean strength of [P], covering occurs.

3. Performance in Two Environments

ZCS represents a simplification of the original classifier system framework in several respects. First, elements of the canonical framework, such as the message list, bid competition, and specificity dependence in bids and payments, were eliminated from the start in order to focus on a more understandable—albeit perhaps in principle less powerful—system. Second, a considerable effort was made to minimize the number of remaining mechanisms while still getting reasonable performance in two test environments. Many plausible mechanisms were experimented with, including explicit generalization pressures, partial matching of conditions against inputs, random classifier injection, and various taxations. Our experiments were not exhaustive, but it appeared that none of these mechanisms produced improvements that justified including them. Conversely, the mechanisms that are included in ZCS appear to be necessary, or, more precisely, the functions they serve in the system should be present one way or another or there will be a loss of performance in the two environments we have studied. In Section 5, we suggest extensions for more complicated environments.

The first test environment, Woods1, is a two-dimensional rectilinear grid containing a single configuration of objects that is repeated indefinitely in the horizontal and vertical directions (Figure 2). The objects are of two types, "food," with sensor code 11, and "rock,"

```
.......................................................
.OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF.
.OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO.
.OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO.
.......................................................
.......................................................
.OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF.
.OOO..OOO..OOO..OOO*.OOO..OOO..OOO..OOO..OOO..OOO..OOO.
.OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO.
.......................................................
.......................................................
.OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF..OOF.
.OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO.
.OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO..OOO.
.......................................................
```

Figure 2. Environment "Woods1" with animat. Empty cells are indicated by ".".

with code 10; blank cells have code 00. The classifier system, here regarded as an animat (Wilson, 1985) or artificial animal, is represented by *. To sense its environment, * is capable of detecting the sensor codes of the objects occupying the eight nearest cells (sensing 00 if a cell is blank). For example, in the position shown, *'s detector input is the 16-bit string 0000000000101011. The left-hand two bits are always those due to the object occupying the cell directly north of *, with the remainder corresponding to cells proceeding clockwise around it. *'s available actions consist of the eight one-step moves into adjacent cells. If a cell is blank, * simply moves there. If the cell is occupied by a rock, the move is not permitted to take place, though one time-step still elapses. If the cell contains food, * moves to the cell, "eats" the food, and receives a reward (r_{imm} = 1000).

An experiment typically proceeded as follows. The classifier population was randomly initialized and then * began executing "problems," each consisting of being placed into a randomly chosen blank cell of Woods1 and moving under control of the system until a food was eaten, at which point the food instantly re-grew and a new problem began. This process was repeated for several thousand problems, with the measure of performance being a moving average over the previous 50 problems of the number of steps (action selections) in each problem. The technique of random restarts with instant re-growing was used to avoid the complication of a nonstationary environment. The bucket brigade was re-initialized ($[A]_{-1}$ set to nil) at each restart since there was no learnable connection between problems.

Curves typical of the best average results achieved by ZCS in Woods1 are shown in Figure 3. The dotted curve represents ZCS as described in Section 2, with parameter values as given in the figure caption. The lower curve is for a slight extension of ZCS, which will be examined in Section 4. Both curves fall very rapidly from a large initial number of time-steps per problem (roughly equal to the average number of time-steps to food for random moves from a random start, which is 27 in Woods1) to about four steps, then descend gradually by another step or so. For comparison, the best that * could possibly do, if from every start it proceeded by the shortest path to the nearest food, would be 1.7 steps. This is the absolute optimum since no more elaborately equipped system having the same actions can do better in Woods1. Thus, the performance achieved by ZCS is very good relative to random, but still roughly twice the absolute optimum.

The primary reason for the shortfall can be seen in Figure 4, which symbolizes, for each distinct position in Woods1, the probability of each of the eight directions of movement at 10,000 problems in a particular run. The length of a line segment in a cell (starting at the cell's center) is proportional to the total strength of the [A] for that direction of movement,

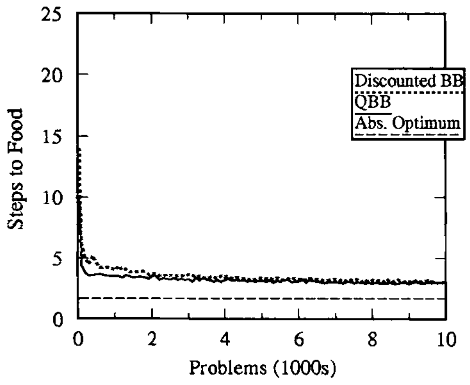

Figure 3. Performance in Woods1. Dotted curve, discounted bucket brigade of Equation 1 (or 2); solid curve, Q-like bucket brigade of Equation 4 (Section 4); dashed curve, absolute optimum. Parameters: $N = 400$, $P_\# = 0.33$, $S_0 = 20.0$, $\beta = 0.2$, $\gamma = 0.71$, $\tau = 0.1$, $\chi = 0.5$, $\mu = 0.002$, $\rho = 0.25$, $\phi = 0.5$. Curves are averages of 10 runs.

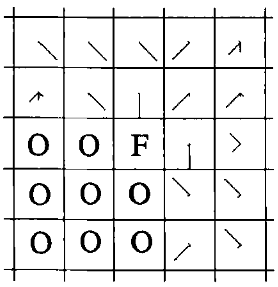

Figure 4. Example of learned move directions in Woods1.

and so to the movement's probability. Note that in many cells the most probable movement dominates the others and points toward the F by the shortest path. An exception is the cell just east of the F, where the dominant move direction is south; the system is apparently blind to the fact that food is just one step west! A possible explanation is that, early in learning, ZCS found and strengthened the northwest move in the cell just southeast of F. This made that cell a source of strong bucket brigade payoff for any move into it. Then, if from the cell just east of F the system happened to discover a southerly move before one to the west, the former move could well come to dominate [M], even if the latter were later generated by the GA. Once the path from the cell east of F became inefficient, the same inefficiency would be passed up along paths leading to that cell, and so on. Thus, the system tends toward "path habits" (Watkins, 1993)—trajectories that due to local payoffs and incomplete exploration, are sub-optimal. The present example is particularly clear, but the effect has been observed to some degree in many experiments. The tendency to path habits may be a consequence at least in part of the exploration/exploitation trade-off implicit in the action-selection and

```
...........O.........................OO.........O.........
.OFO.......F........F.........O.......F.........FO........
...........O........OO.......F............................
............................O......O.........F......O.....
...F.......OFO........OFO..........F.........OO......F....
...OO..............................O..................O...
..........OO.......O........OO....................O.......
.OFO.......F.......OF........F.........OFO........F.......
..................................................O.......
...OO.......O.................O........OO.......O.........
...F.......F.......O.........FO........F........OF....OFO.
...........O.......OF.....................................
..O......................O............O......O.......O....
..F.......F...............F..........FO.......F......OF...
..O.......OO...............O...................O..........
.....................O.....................O..............
...F........OFO......F..........F.........F......OF.......
...OO................O..........OO.......O.......O........
```

Figure 5. Environment Woods7.

reinforcement regimes. In particular, a large value of τ causes any reasonably good move to quickly dominate [M] and tends to prevent its being replaced by a better one. This may be desirable for fast learning but can limit maximum performance, as observed here in Woods1.

ZCS was tested in a second environment, Woods7, shown in Figure 5 (the top and bottom edges of Woods7 are connected, as are the left and right edges). Though they have the same object types, the two environments differ fundamentally in that, viewed as a finite state machine taking inputs (actions) and producing outputs (sensations and rewards), Woods1 is Markovian with delayed rewards. Woods7, on the other hand, is non-Markovian with delayed rewards. In the terminology of Wilson (1991), they are Class 1 and Class 2 environments, respectively. The Markovian property means in this context that given any sensory input x, the sensory input y (and the reward) resulting from taking an action a is always exactly predictable. Inspection will show this holds for Woods1 but not for Woods7. Stated somewhat differently, to know one's position in Woods1 (with respect to the basic configuration), it is sufficient to know the current input. In Woods7, however, it is necessary either to see more than one step away or to remember some recent sensory inputs. Since ZCS has no temporary memory, and its view in these experiments is limited to adjacent cells, ZCS cannot in general be expected to gain food in Woods7 at the rate possible for a system without these restrictions (Littman, 1993). However, it is still of interest to see how well ZCS can do.

A simple routine that marks every open cell with its minimum distance from food and then averages the distances reveals that the absolute optimum performance (again defined as the best possible performance of a creature having arbitrary memory and sensory capability) under the random restart regime is 2.2 steps in Woods7. On the other hand, moving randomly until food is encountered averages 41 steps. ZCS's performance as shown by the two upper curves in Figure 6 is about 5 steps after a few thousand problems.[1] Thus, here, as in Woods1, ZCS is very good with respect to random, but still about twice the absolute optimum. However, ZCS's shortfall may be for somewhat different reasons in the two cases. In Woods1, it was due to path habits; with better exploration, ZCS might well have reached the absolute optimum, since Woods1 is a Class 1 environment and the absolute optimum

1 For comparison, the original animat's (Wilson, 1985) performance on Woods7 was asymptotically about equal to ZCS's, but it learned faster due to its look-ahead classifier creation heuristic.

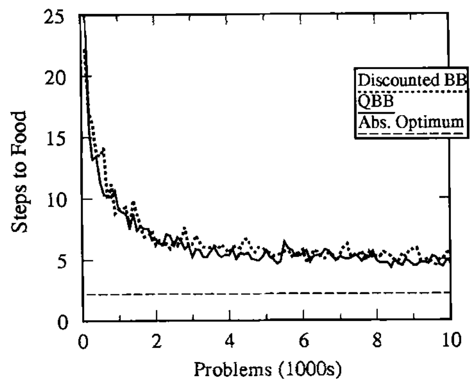

Figure 6. Performance in Woods7. Dotted curve, discounted bucket brigade of Equation 1 (or 2); solid curve, Q-like bucket brigade of Equation 4 (Section 4); dashed curve, absolute optimum. Parameters the same as in Figure 3. Curves are averages of 10 runs.

is achievable by a memoryless system. In Woods7, performance was probably also limited because Woods7 is Class 2 and, thus, can't be "solved" without temporary memory (or more elaborate sensors). Qualitatively, ZCS exhibits considerable drive toward food in Woods7: when nonblank objects enter its sensory field, it either takes food immediately if adjacent or tends to move around a rock to get the food on the other side as though it "knows it's there." When surrounded by blanks, ZCS tends to drift in a general direction that is maintained through several problems (for an explanation, see Cliff & Bullock, 1993). However, upon encountering a rock or rock pair, ZCS sometimes goes the long way around to reach food or may even bang the rock, showing a residual uncertainty apparently because with its lack of temporary memory it can't resolve which of two or more situations it is actually facing. We believe the shortfall in performance is due at least in part to this inevitable uncertainty in Woods7, and not solely to the formation of path habits.

Ideally, in judging ZCS's performance we would like to compare it with an optimal temporary memoryless system. That is, how well could a system do which, say, had a map of Woods7 but was prohibited from remembering any past inputs or outputs? Perhaps surprisingly, this calculation is difficult because the ambiguity of sensory inputs introduces the possibility of loops, which can apparently be avoided only by stochastic behavior or elaborate planning. In fact, the problem of calculating the optimal performance of a memoryless, sensory-limited system in a Class 2 environment is NP-complete (Littman, 1994). However, in the same paper Littman goes on to use a branch and bound technique to find and prove optimal a memoryless policy—in effect a set of classifiers—that achieves a performance of 3.10 steps in Woods7, leaving learning classifier systems like ZCS appreciable room for improvement.

4. ZCS and Q-learning

The performance and reinforcement cycles of ZCS have strong resemblances to Q-learning (Watkins, 1989), perhaps the most widely used reinforcement learning algorithm. To see the connection it is helpful to return to (1) and rewrite it in Schwartz's (1993) notation using the update operator $\overset{\beta}{\leftarrow}$ defined for scalar variables x and y by

$$x \overset{\beta}{\leftarrow} y \equiv x \leftarrow x + \beta(y - x)$$

Note that $\overset{\beta}{\leftarrow}$ is just the Widrow-Hoff (1960) learning procedure, where x is corrected at learning rate β by an error $y - x$. In terms of $\overset{\beta}{\leftarrow}$, expression (1) can be written

$$S_{[\mathrm{A}]} \overset{\beta}{\leftarrow} r_{imm} + \gamma S_{[\mathrm{A}]'} \tag{2}$$

which says that the reinforcement cycle adjusts the total strength of each action set to estimate immediate reward plus the discounted strength of the succeeding action set.

In Q-learning, an evaluation function estimate $\hat{Q}(x, a)$ is updated at each time-step. Here x represents the current sensory input and a represents an output action from the set of available actions, A. The update operation can be written

$$\hat{Q}(x, a) \overset{\beta}{\leftarrow} r_{imm} + \gamma \max_{a' \in A} \hat{Q}(x', a') \tag{3}$$

where x' is the input on the next time-step. It has been proved (Watkins, 1989) that if the environment is Markovian (Class 1) and all x, a combinations are tried sufficiently often, this update procedure will cause $\hat{Q}(x, a)$ to converge to a function $Q(x, a)$ such that by carrying out the action that maximizes $Q(x, a)$ for each x, the system will optimize the discounted sum of future rewards, $\sum_{j=0}^{\infty} \gamma^j r_{t+j}$, that it receives. Thus if the discounted sum is accepted[2] as a reasonable measure of return—it rates rewards received sooner higher than equal rewards received later, and so emphasizes expeditious behavior—and if the environment is Class 1 and sufficiently explored, Q-learning offers a technique for achieving optimal performance with the advantage of being quite well understood analytically. Similar proofs do not exist for Class 2 environments, nor for the case where the set of possible inputs x is so large that explicit estimation of $Q(x, a)$ for every x is impractical and the technique must be modified to generalize over "similar" xs. Nevertheless, Q-learning can work quite well in practice, as shown for instance in Lin (1993). There, a neural network, which generalized over inputs, learned a Q-like function in a Class 2 environment and performance was good.

If we compare (2) and (3) we see that they are similar except for the max operation in (3). Consider the match set $[\mathrm{M}]'$, and recall that in general it contains classifiers advocating various actions; that is, it contains several different potential action sets, one for each advocated action. We could imagine modifying the update rule in (2) as follows:

$$S_{[\mathrm{A}]} \overset{\beta}{\leftarrow} r_{imm} + \gamma \max_{[\mathrm{A}]' \subseteq [\mathrm{M}]'} S_{[\mathrm{A}]'} \tag{4}$$

where now the max means the potential action set in $[\mathrm{M}]'$ with the highest total strength. (4) would then closely parallel the Q-learning rule, Expression (3). The difference is, briefly, that whereas Q-learning estimates a function of input-action combinations, the modified bucket brigade of (4) updates *rule set*-action combinations. While $S_{[\mathrm{A}]}$ is not obviously a function in the sense $Q(x, a)$ is, we can gain perspective by considering that for every input x and action a, the system will, in general, produce an action set [A], with strength $S_{[\mathrm{A}]}$, whose members all match x and advocate a. Thus, the current population [P] can be regarded as embodying a *mapping* from X $\times$ A to strengths S. As noted in Section 2, the mapping is capable of

2 See Schwartz (1993) for a reinforcement algorithm that is not based on discounting future payoffs.

generalization because, due to #s in the classifiers, a given [A] will often match more than one x. While we have no proof that what we shall call the *Q-bucket brigade* (QBB) of (4) leads to a population that maps optimally, we can still compare its experimental performance to that of better understood relatives like Q-learning.

To do this, let us first explain that the solid curves in Figures 3 and 6 were obtained via the QBB update rule (4), whereas the dotted curves are for rule (2), without the "max." Q-learning was tried informally on Woods1 by Kaelbling (personal communication, 1993), who employed her *interval estimation* (Kaelbling, 1990) exploration technique. The result was very rapid descent in steps to food to very nearly the optimum, faster and better than ZCS using QBB. On Woods7, the comparison was closer. Kaelbling's system reached approximately four steps to food on average (vs. five for ZCS), somewhat more rapidly. Littman (1993) reports "Q-learning identified a Class 1 agent that averages 0.27 foods per step [3.70 steps to food]"—i.e., this was the best performance of several runs. Both Kaelbling and Littman implemented the Q function as a lookup table. We interpret the results of these informal and preliminary comparisons as follows. On Woods1, Q-learning performed essentially perfectly, as it should, because the environment is Class 1 and effective exploration occurred. Our hypothesis is that ZCS performed less well on Woods1 because the exploration technique was less effective and, in particular, had no way of exploring action sets not represented in [M]. On Woods7, the comparison is much closer, with performance for both Q-learning and ZCS limited primarily by the fact that Woods7 is Class 2. We hypothesize that ZCS's more primitive exploration technique accounts for the performance difference.

Besides ZCS, there are Q-learning-like mechanisms in Roberts's (1993) classifier system Dyna-Q-CS, which is designed to do dynamic planning along the lines of Sutton's (1991) DYNA architecture. Dynamic planning requires that the system keep track of the sensory input consequent to an action taken in a given state; from this information the system can perform Q-value updates on hypothetical as well as actual events. Because classifiers do not encode or predict the input that will follow their activation, Roberts introduced a bounded data structure called a *followset*, associated with each classifier, to record this information as it is gained. While Dyna-Q-CS appears to be a promising approach to model building in classifier systems, updates using the followsets are intricate. In addition, Dyna-Q-CS's results on Woods7—21.7 steps to food asymptotically—suggest the system is not yet fully shaken down.

5. Extensions to ZCS

Having in the last few sections described ZCS, illustrated its performance, and shown its relation to Q-learning, we now go on to suggest ways in which ZCS, as a basic, understandable classifier system foundation, can be extended. For this we will take the foundation to be as described in Section 2, except that instead of the update expression (2) we will from now on define ZCS to use the more completely Q-like expression (4).

5.1 Temporary Memory

In Holland's original framework, information could in principle be carried over from one time-step to the next by posting a message to the internal message list, then matching it on the next cycle. Unfortunately, possibly for reasons of formal simplicity, internal messages were defined as strings of length equal to strings from the detectors, so that the internal messages tended to be long. Temporary memory and other uses of messages depend on classifiers that post messages that other classifiers can match. Unless special non-GA triggered operators are employed (Riolo, 1989), evolution of appropriately coupled classifiers becomes increasingly improbable as the messages grow longer. In fact, there appears to be only one investigation in the literature (Smith, 1991) reporting significant generation of couplings by the GA alone, and there the messages were short.

We sought a simple extension to ZCS that could permit transfer of information between time-steps but with a minimum of mechanism so that evolution of appropriate classifiers would occur with reasonable probability. At the same time, it seemed that in many problems of interest, the amount of information that needed to be transferred was quite small. For instance, often a decision depends simply on whether or not some prior event has occurred, i.e., just one bit may suffice. Accordingly, we propose to extend ZCS using a memory-register approach. For this, consider an expanded classifier format of which the following is an example:

#100,11# : 01,##0.

Here, the condition has two parts, an environmental part #100, and a memory register part, 11#, both of which must match for the condition to be satisfied. The memory register itself is a three-bit register whose contents are set and changed by a supplement to the action side of the classifier, as follows. The first part of the action side, 01, is an action (including the null action) affecting the external world. The new second part, ##0, codes an internal action affecting the memory register: a 0 or 1 writes that value to the corresponding register position, a # leaves its corresponding position alone. (In a variant, there would be just one action part coding either an external action or an internal one affecting the register.)

The simplest experiment would implement a one-bit register in an environment designed so that exactly one bit of temporary memory would be important in certain situations. If the GA indeed took advantage of the register, more elaborate experiments with a progressively larger register could be tried. An interesting possibility would be that, having found some register bits useful, the system would evolve classifiers referring not only to those bits but to others representing refinements of the information in the first bits. Certain positions of the register might in this speculation evolve to denote tags that labeled behavioral modules, or intentions, with other bits controlling the more detailed action of the modules, in effect implementing a behavior hierarchy (Wilson, 1987b). Of course, all such outcomes imply extensive further research (including additions to the performance and reinforcement cycles), but we suggest at this point that a memory register approach, due to its simplicity, is at the moment a better test of classifier systems' ability to implement temporary memory than the complex message list of the canonical framework. Quite separately, the approach forms an interesting contrast to the use of recurrent neural networks to implement temporary memory in reinforcement learning (Elman, 1990; Lin, 1993). In the latter, effective "history features" are generated by unfolding the recurrent net and training via back propagation. In the ZCS memory register approach, potential history features take the form of bits placed in the register that are matched by succeeding classifiers. Then features that indeed matter for performance would cause selection of the classifiers that they couple. The difference between the two approaches to temporary memory is a difference between learning by error correction and learning by a Darwinian process of variation and selection.

5.2 Performance-Based Action Selection

ZCS uses the relatively primitive roulette wheel action selection described in Section 2. The probability of selection of an action a_i, $Prob(a_i)$, is proportional to the total strength $S_{[M]_i}$ of classifiers in [M] advocating a_i. A more flexible method is to let

$$Prob(a_i) = \left(\exp\left(S_{[M]_i}/T\right)\right) / \left(\sum_k \exp\left(S_{[M]_k}/T\right)\right),$$

then let T be large at the start of a run where more exploration is presumably desirable, and reduce it with time in order to make the judged best actions more probable. Still, the "annealing schedule" $T(t)$ must be determined in advance, unrelated to the system's actual success or lack thereof. In the reinforcement learning literature, a class of more sophisticated methods keeps track of the variance or uncertainty of Q (say) estimates and explores with the aim of progressively reducing uncertainty but with a bias toward apparently better actions. Kaelbling's interval estimation is one such method, in which $Prob(a_i)$ is proportional to the sum of the Q estimate and its variance. While effective in many problems, the uncertainty reducing methods can still miss achieving optimality if the parameters of the explore/exploit balance are not within appropriate ranges. We wondered whether these essentially formal methods could benefit from a perspective that took into account the system's overall performance or prosperity. For other work that takes a step in this direction see Lin (1993) and Dorigo (1993).

Consider a system that integrates rewards over time, subtracting its metabolic costs to yield a current energy reserve $E(t)$. The system might be initialized with a reserve E_0. Would it not make sense for the system's explore/exploit balance to depend on $E(t)$? For instance, the probability of exploration *Prob(explore)* could be given by a function $f(E(t))$ where f would have been set by evolution. For high values of E, there could be a strong tendency to explore, corresponding to play or curiosity in an animal. A large value of E_0 would then mean considerable exploration while young, yielding to greater exploitation as $E(t)$ fell and the system had to gain its own food. Later, through learning, $E(t)$ could start rising, leading eventually to prosperity that would again permit exploration or play. Of course, this picture is too simple: if the system got in trouble and $E(t)$ began falling precipitously, greater exploration might suddenly be in order. Whatever the characteristic forms of *Prob(explore)*, the point is that the system's overall success is relevant to action selection in a way that deserves study. One approach we are investigating is to let the T in the exponential roulette wheel above depend on $E(t)$ through various plausible functions f. This can be applied straightforwardly to ZCS with the slight addition that actions a_i such that $[M]_i$ is empty should still be represented, with low strength values, in the probability calculation. Then, if such an action is chosen, a classifier would be created to execute it. In this way ZCS could break out of path habits for cases where the best action is not represented in [M].

5.3 Niche GA

The genetic algorithm employed in ZCS's discovery component is *panmictic*: a classifier has an equal probability of mating (crossing) with any other classifier in [P] having a given strength. The GAs used in function optimization problems are usually panmictic (with the notable exception of parallel implementations where a notion of geographic distance between strings may control the mating probability). It is felt that in problems such as optimization where a single best point is sought, samples from all parts of the space should be capable of recombination. The situation is different, however, for spaces being searched by classifier systems. There, solving the problem usually means solving a set of subproblems each due to an aspect of the system's environment. Thus the GA in a classifier system faces a multiple optimization problem in which it must allocate resources—classifiers—to search more or less separate niches of the space. The situation is further complicated by the fact that, often, success in a niche is contingent on success in another niche. For instance, this occurs if, as in ZCS, classifiers form chains to reach reward: each step in the chain can be regarded as an environmental niche for which good classifiers must be found; if not, the chain as a whole is weakened or broken. Under such circumstances it is not obvious that a panmictic GA is the best way to identify and search the niches, since it is not clear that the recombination of classifiers from an arbitrary pair of niches will be relevant to either of them or to any other niche.

Quite separately from mate choice, the practice in a panmictic GA of selecting the first parent from the population as a whole, or globally, is problematic for classifier systems in which a classifier's fitness is based on strength. Different niches may quite normally have different reward levels and, therefore, different strengths in the corresponding classifiers. If the niches are separate and unrelated, it makes sense to base selection on strength because then greater resources will go to the more remunerative niches. However, if the niches belong to the same chain, and the reinforcement cycle has a discount factor $\gamma < 1$, then early niches will necessarily have less strength than later ones. It is not clear that the early niches should get fewer resources, though, since they "set up" and so permit reward ultimately to be received in the later ones.

The classifier system BOOLE (Wilson, 1987a) successfully used a panmictic GA to learn the Boolean multiplexer, a highly nonlinear logical function. There, however, the niches, which corresponded to disjuncts of the function, had equal reward schedules, dependent only on whether the system was right or wrong, and the problem was not sequential. In addition, the environment was "dense" in the sense that the system would see every possible input string eventually. That meant that every classifier produced by panmictic crossover would eventually match and receive an evaluation, so that every cross was in some degree fruitful and contributed to the search.

In contrast, environments like Woods1 and Woods7 are "sparse" in the sense that the set of inputs that occur in them form a very small subset of the possible strings under that encoding. Consequently, panmictic crossover in ZCS often produces offspring that never match and so remain dead wood in [P] until deletion happens to eliminate them. This restricts the population computation space—the set of evaluable classifiers—and does not contribute to the search for better classifiers in the niches that exist. While ZCS is largely successful in Woods1 and Woods7, it is possible that a nonpanmictic niche-based GA would improve performance. Booker (1982, 1989) has investigated and long advocated a niche GA for classifier systems, for many of the reasons noted here. His implementation is part of a larger system having many aspects not present in ZCS, and to keep to ZCS's spirit of minimalness we propose here a niche GA that contains some but not all of the mechanisms in Booker's implementation. One of his insights was that niches are effectively identified by the various match sets that occur, so that a GA designed to search the niches should be restricted to classifiers in [M]. A further suggestion was that if, under exact matching, [M] was empty or small, classifiers that nearly or partially match the input should be allowed to participate in the niche GA. The idea was that partial matchers contain some, albeit imperfect, information relevant to the niche and so should participate if there were few exact matchers. We are unsure of the value of using partial matches, since the mismatches are a form of noise, and we leave this aspect out for now.

In line with Booker's basic concept, we propose an extension to ZCS in which a niche GA supplements the existing panmictic one. The niche GA would simply act on [M] with a certain probability ρ_{niche}; there would be no special triggering conditions. The GA would operate as usual, except that if [M] contained just one classifier, only mutation and not crossover could occur. Deletion would occur from [P] as a whole, using a variation on a niche balancing technique that Booker developed. We would have each classifier keep an estimate (by exponential averaging) of the number of classifiers in its match sets. Then a classifier's probability of deletion would be proportional to this quantity, so that niches would tend to have the same number of members, independent of niche strength. This avoids the panmictic GA's problem of giving more resources to niches later in a chain. Finally, the panmictic GA would be retained, but at a reduced probability of invocation, to offset any inbreeding tendency of the niche GA, and to give some degree of emphasis to clearly more remunerative niches. Because the niche GA crosses classifiers in the same [M], the offspring are guaranteed to match the same input, so the generation of never-matching classifiers in sparse environments should be greatly reduced. An experimental classifier system has been developed along the above lines (Wilson, in preparation).

5.4 S-classifiers

As discussed in Section 4, ZCS's population [P] at any moment constitutes a mapping from x, a pairs to action set strengths $S_{[A]}$ that control the system. The objective of the reinforcement and discovery components is to evolve mappings [P] that produce better and better performance in the environment. In this section we consider the mapping's underlying units, the classifiers, and how their representation might usefully be modified. Classifiers as we usually know them are somewhat limited beasts in that their conditions are expressed as conjunctive predicates. That is, they match, or are true, if and only if all of the bits specified in the condition match. A classifier can express a generalization, but only in a particular way: by saying "these bits must be just so, and the rest don't matter," i.e., the generalization must be a hyperplane. It is not possible, in a single conjunctive classifier, to express a generalization such as "bit 3 or bit 5 must be a one," or "at least two of bits 3, 5, and 7 must be ones." Of course, binary classifiers completely ignore the world of continuous variables. Some work (e.g., Grefenstette, 1992) has been done with classifiers whose conditions are specialized for certain domains, and, as mentioned earlier, there is work on continuous inputs via fuzzy classifiers. However, there is only a little work (Booker, 1991) toward a general reformulation of classifier representation that would permit expression of arbitrary generalizations. The main reason has been that it was not clear how to apply genetic operations to arbitrary logical expressions, whereas one did see how to do it for conjuncts expressed in the 1, 0, # language. Now, however, the development of genetic programming (Koza, 1992) has shown how genetic operations can be applied successfully to arbitrary functions encoded as LISP s-expressions. It is straightforward to represent a classifier's condition as an s-expression built from `and`, `or`, and `not`. Other operators such as a thresholded sum and conditional comparison can be included in the system's function set, so that very flexible conditions can be evolved with a set of primitives that is still quite small. An "s-classifier" would thus have an s-expression as its condition; its action would be a bit string as usual (though one can imagine the action also as an s-expression). The advantage of s-classifiers over standard ones would be to permit the evolution of generalizations that suit the $S_{[A]}$ functions of arbitrary environments better than can the standard classifier's conjunctive syntax.

6. Summary and Conclusions

This paper has presented ZCS, a basic classifier system for reinforcement learning that retains much of Holland's original framework while simplifying it so as to increase understandability and performance. ZCS's relation to Q-learning was brought out, and its performance shown to be quite similar in two experimental environments. Four extensions of ZCS were proposed: temporary memory, more sophisticated action selection, a niche GA, and "s-classifiers."

There would appear to be several principal advantages of a classifier system like ZCS in reinforcement learning problems. First, ZCS is a Q-like system that has its own built-in basis for generalization—the classifiers themselves. This contrasts with lookup table implementations of Q-learning, which either don't generalize or must be supplemented by clustering methods (Mahadevan & Connell, 1992), and with neural-network-based Q-learners that generalize through the sometimes slow process of backpropagation. In addition, the proposed s-classifier extension to ZCS may permit achievement of efficient generalizations with respect to the underlying space. Second, ZCS appears to be extensible to handle environments calling for temporary memory and hierarchical behavior using a memory register technique not seen in other approaches to reinforcement learning. Finally, like any classifier system, ZCS offers a Darwinian perspective on reinforcement learning that contrasts with the many approaches that rely principally on error correction.

With respect to Holland's original framework, the main element greatly lost sight of in ZCS is, of course, the message list and the concomitant potential for multiple parallel thought lines and the formation of complex mental models. One sees in principle how these things could emerge in a classifier system, but we don't yet understand the mechanics well enough to make it happen. We hope that further research on simpler systems like ZCS will contribute to bringing the full system to maturity.

Acknowledgments
I am grateful for extensive interactions on most of these topics, in person and by electronic mail, with Lashon Booker and Rob Smith. I also wish to thank Leslie Kaelbling and Michael Littman for helpful discussions. Comments by three anonymous reviewers, as well as by Littman and Marco Dorigo, made important differences in the final version.

References

Barto, A. G. (1992). Reinforcement learning and adaptive critic methods. In D. A. White & D. Sofge (Eds.), *Handbook of intelligent control: Neural, fuzzy and adaptive algorithms* (pp. 469–491). New York: Van Nostrand Reinhold.

Booker, L. B. (1982). *Intelligent behavior as an adaptation to the task environment.* Ph.D. dissertation, The University of Michigan, Ann Arbor.

Booker, L. B. (1989). Triggered rule discovery in classifier systems. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 265–274). San Mateo, CA: Morgan Kaufmann.

Booker, L. B. (1991). Representing attribute-based concepts in a classifier system. In G. J. E. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 115–127). San Mateo, CA: Morgan Kaufmann.

Cliff, D., & Bullock, S. (1993). Adding "foveal vision" to Wilson's animat. *Adaptive Behavior, 2*(1) 49–72.

Dorigo, M. (1993). Genetic and non-genetic operators in ALECSYS. *Evolutionary Computation, 1*(2), 151–164.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14*, 179–211.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Grefenstette, J. J. (1992). The evolution of strategies for multiagent environments. *Adaptive Behavior, 1*, 65–90.

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

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. Volume II.* Los Altos, CA: Morgan Kaufmann.

Kaelbling, L. P. (1990). *Learning in embedded systems.* Ph.D. dissertation, Stanford University, Stanford, CA.

Koza, J. R. (1992). *Genetic programming.* Cambridge, MA: The MIT Press/Bradford Books.

Lin, L.-J. (1993). *Reinforcement learning for robots using neural networks.* Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA.

Littman, M. L. (1993). An optimization-based categorization of reinforcement learning environments. In J.-A. Meyer, H. L. Roitblat, & S. W. Wilson (Eds.), *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (pp. 262–270). Cambridge, MA: MIT Press/ Bradford Books.

Littman, M. L. (1994). Memoryless policies: theoretical limitations and practical results. In D. Cliff, P. Husbands, J.-A. Meyer, & S. W. Wilson (Eds.), *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior.* Cambridge, MA: MIT Press/Bradford Books.

Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence, 55*, 311–365.

Parodi, A., & Bonelli, P. (1993). A new approach to fuzzy classifier systems. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 223–230). San Mateo, CA: Morgan Kaufmann.

Riolo, R. L. (1989). The emergence of coupled sequences of classifiers. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 256–264). San Mateo, CA: Morgan Kaufmann.

Roberts, G. R. (1993). Dynamic planning for classifier systems. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 231–237). San Mateo, CA: Morgan Kaufmann.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Machine Learning: Proceedings of the Tenth International Workshop.* San Mateo, CA: Morgan Kaufmann.

Smith, R. E. (1991). *Default hierarchy formation and memory exploitation in learning classifier systems.* Ph.D. dissertation, The University of Alabama, Tuscaloosa, AL.

Sutton, R. S. (1991). Reinforcement learning architectures for animats. In J.-A. Meyer & S. W. Wilson (Eds.), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior* (pp. 288–296). Cambridge, MA: MIT Press/ Bradford Books.

Sutton, R. S. (1992). Introduction: The challenge of reinforcement learning. *Machine Learning, 8*, 225–227.

Valenzuela-Rendón, M. (1991). The fuzzy classifier system: a classifier system for continuously varying variables. In R. Belew & L. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 346–353). San Mateo, CA: Morgan Kaufmann.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* Ph.D. dissertation, Cambridge University, Cambridge, UK.

Watkins, C. J. C. H. (1993). Q-learning. Talk presented at the workshop "Reinforcement Learning:

What We Know, What We Need," June 30 and July 1, 1993, University of Massachusetts, Amherst, MA.

Widrow, B., & Hoff, M. (1960). Adaptive switching circuits. *1960 IRE WESCON Convention Record* (pp. 96–104). New York: IRE.

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). Hillsdale, NJ: Lawrence Erlbaum Associates.

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. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 217–220). Los Altos, CA: Morgan Kaufmann.

Wilson, S. W. (1988). Bid competition and specificity reconsidered. *Complex Systems*, *2*(6), 705–723.

Wilson, S. W. (1991). The animat path to AI. In J.-A. Meyer & S. W. Wilson (Eds.), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior.* Cambridge, MA: MIT Press/Bradford Books.

Wilson, S. W. (in preparation). Action-selection and generalization over payoff landscapes.

Wilson, S. W., & Goldberg, D. E. (1989). A critical review of classifier systems. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 244–255). San Mateo, CA: Morgan Kaufmann.

This article has been cited by:

1. Matthew Studley, Larry Bull. 2007. Using the XCS Classifier System for Multi-objective Reinforcement Learning Problems. *Artificial Life* **13**:1, 69-86. [Abstract] [PDF] [PDF Plus]
2. Jacob Hurst, Larry Bull. 2006. A Neural Learning Classifier System with Self-Adaptive Constructivism for Mobile Robot Control. *Artificial Life* **12**:3, 353-380. [Abstract] [PDF] [PDF Plus]
3. Tim Kovacs . 2004. Rule Fitness and Pathology in Learning Classifier Systems. *Evolutionary Computation* **12**:1, 99-135. [Abstract] [PDF] [PDF Plus]
4. Larry Bull , Jacob Hurst . 2002. ZCS Redux. *Evolutionary Computation* **10**:2, 185-205. [Abstract] [PDF] [PDF Plus]
5. Andy Tomlinson , Larry Bull . 2001. Symbiogenesis in Learning Classifier Systems. *Artificial Life* **7**:1, 33-61. [Abstract] [PDF] [PDF Plus]
6. Pier Luca Lanzi , Stewart W. Wilson . 2000. Toward Optimal Classifier System Performance in Non-Markov Environments. *Evolutionary Computation* **8**:4, 393-418. [Abstract] [PDF] [PDF Plus]
7. Eric B. Baum , Igor Durdanovic . 2000. Evolution of Cooperative Problem Solving in an Artificial Economy. *Neural Computation* **12**:12, 2743-2775. [Abstract] [PDF] [PDF Plus]
8. Stewart W. Wilson. 1995. Classifier Fitness Based on Accuracy. *Evolutionary Computation* **3**:2, 149-175. [Abstract] [PDF] [PDF Plus]