

Modifying XCS for Size-Constrained Systems

Devon Dawson

Hewlett-Packard, Roseville
8000 Foothills Blvd.
Roseville, California 95747
Tel. 916 785-2688
Fax. 916 785-1199
devon_dawson@hp.com

Benjoe Juliano, Ph.D.

Department of Computer Science
California State University, Chico
Chico, California 95929-0410
Tel. 530 898-4619
Fax. 530 898-5995
Juliano@ecst.csuChico.edu

Abstract

Extended Classifier Systems, or XCS, is a soft-computing approach to machine learning in rule-based systems. While XCS has been shown effective in learning accurate, compact and complete mappings of an environment's payoff landscape, it can require significant resources to do so. This paper presents four modifications that allow XCS to achieve high performance even in highly size-constrained populations. By modifying the genetic algorithm trigger function, the classifier deletion-selection mechanism, the frequency of classifier parameter updates and the genetic algorithm selection function – the modified system more efficiently uses the available population resources. Experimental results demonstrate the improvement in performance achieved with the proposed modifications in both the single-step 6-Multiplexer problem and the multi-step Woods-2 problem.

Keywords: XCS, extended classifier systems, size-constrained systems.

1. Introduction

Classifier Systems (CS) are a soft-computing approach to machine learning in rule (*i.e.*, classifier) based systems first introduced by Holland [1,2]. While CS showed some early promise, the use of *strength* (*i.e.*, accumulated reward) as the basis for a classifier's fitness appeared to hinder the system's learning ability, particularly in environments requiring biased payoff landscapes and long chains of sequential classifiers [3]. Subsequently, Wilson [3] introduced his *accuracy*-based Extended Classifier System (XCS), in an attempt to address the shortcomings exhibited by traditional strength-based CS.

XCS uses the accuracy of classifier payoff predictions as the sole measure of a classifier's fitness. By using an accuracy based fitness measure independent of the magnitude of the payoff predicted, as well as a *niched* Genetic Algorithm (GA) [4], XCS has demonstrated an implicit tendency towards developing and maintaining a complete and accurate mapping of $X \times A \Rightarrow P$ [3]. That is, XCS strives to learn a complete mapping from system inputs (X) to classifier actions (A) to predicted environmental payoff (P) should the given action be taken in that input state. Furthermore, XCS appears to work towards the development of maximally general and optimal classifiers [3,5], which are classifiers that cover the maximum number of possible input conditions while still remaining accurate in their payoff predictions.

While the original XCS definition has been shown to be able to achieve high performance in a variety of single step and multi-step problems [3,5], the resources (*i.e.*, population/memory size and machine cycles) required to develop and maintain a complete map of all but the most basic payoff landscapes can be significant.

In an attempt to allow XCS to scale more effectively to larger and more complex problems, the authors present four modifications that help XCS learn accurate and effective populations of classifiers in systems with significantly smaller population sizes than is possible with Wilson's original XCS definition. Experimental results demonstrate the improvement in performance achieved with the proposed modifications in both single-step 6-Multiplexer and multi-step Woods-2 problems [3].

2. Classifier Systems

With the introduction of his XCS [3], Wilson argued that many of the difficulties encountered by traditional CS were owing to their use of strength as the measure of classifier fitness. Building upon his earlier work on what he termed a simplified “zeroth-level” classifier system (ZCS) [6], Wilson’s XCS simplified the internals of traditional CS in an effort to allow a clearer picture of the processes involved.

Due to the combination of an accuracy-based fitness measure and a niched [4] Genetic Algorithm (GA), which operates on functionally related sets of classifiers rather than *panmictically* (i.e., on the entire population), XCS was shown to be capable of learning accurate and complete mappings of an environment’s payoff landscape [3,5].

This section describes the fundamental components and processes involved in the definition and execution of XCS. For a detailed description, the reader is referred to the literature [3,5,7].

2.1 Classifier System Overview

The execution of XCS typically proceeds as follows:

- 1) An input message is sensed by the detectors;
- 2) Classifiers are compared against the input message with those whose conditions are satisfied flagged as candidates for activation;
- 3) The *Conflict-Resolution* component is invoked to select one of the actions represented by the satisfied classifiers;
- 4) The activated classifier’s action (*i.e.*, message) is placed on the system output;

- 5) The *Credit-Distribution* component updates the parameters of the relevant classifiers; and
- 6) When a given threshold is met, the Genetic Algorithm is invoked.

2.2 Classifier System Components

2.2.1 Messages

Messages in a CS are traditionally composed of fixed length bit strings defined over the binary alphabet $\{0,1\}$. These messages are used to encode the current state of the system, as sensed by the “detectors.”

2.2.2 Classifiers

Classifiers in a CS are simple condition / action rules. The condition part of a classifier consists of one or more bit string masks (more specifically, two were used in the original work by Holland [1,2], while XCS uses a single condition per classifier [3]). Differing slightly from the binary alphabet of messages, conditions are defined over the ternary alphabet $\{0,1,\#\}$ – where # implies a “Don’t-Care” bit in the mask. A don’t-care symbol at a particular position indicates that the condition may match any value in the same position of any given message string.

A classifier’s condition is considered satisfied if the current input message matches the condition’s mask string. For example, a classifier with a condition of $\{1,\#,0\}$ would be satisfied if either message, $\{1,0,0\}$ or $\{1,1,0\}$ were present. When the condition of a classifier is satisfied, it is considered a candidate for activation by the Conflict-Resolution component.

The action portion of a classifier is another bit string defined over the binary language of $\{0,1\}$. If chosen for activation by the Conflict-Resolution component (described below), the classifier's action is placed on the system output.

2.2.3 Conflict-Resolution

The Conflict-Resolution component in a CS is responsible for determining which of the candidate rules in the match-set (*i.e.*, the set of rules whose conditions were matched by the current input) will be selected for activation. Because of wanting to both explore the problem space and test the performance of the system learned so far, XCS uses two forms of conflict-resolution depending on the nature of the current iteration, called “exploration” and “exploitation” [3,12].

During exploration episodes, the system selects an action at random from the group of candidate actions for activation. During exploitation episodes, the sets of classifiers advocating the various actions (*i.e.*, action sets) are ranked based on the fitness weighted payoff prediction of each action set and the set with the highest weighted prediction is activated. In this way, the system's current “best-guess” is activated and its performance can be measured.

XCS determines whether a conflict-resolution episode is an exploration or exploitation episode by choosing between the two possibilities with an equal probability; other techniques may be beneficial [12].

2.2.4 Credit-Distribution

The method used by XCS to update a classifier's predicted payoff is modeled after the reinforcement-learning technique of *Q-Learning* [8,9]. Q-Learning has been found to be effective in learning an estimate of the external reward anticipated on some

subsequent time step from a classifier's activation on the current time step. The credit-distribution algorithm in XCS acts to propagate predicted rewards "across" the activation chain, discounting them at each intervening time step by a constant fraction, γ . It was Wilson's goal to use Q-Learning to replace the traditional "bucket-brigade" algorithm of Holland's CS [1] with a more theoretically sound technique.

While the bucket-brigade made an effort to transfer credit from message consumers to producers, the Q-Learning technique implemented in XCS merely assigns the sum of any external reward received on the previous clock cycle and the maximum payoff predicted on the current clock cycle discounted by γ .

2.2.5 Genetic Algorithm

The Genetic Algorithm (GA) component of a CS provides the searching behavior of the system using a process modeled after Darwinian natural selection. GAs were first introduced by Holland [10] as a framework with which to test adaptation in genetically modeled systems.

An evolutionary search as implemented by a GA is a relatively simple process involving the following four steps:

- 1) *Selection*: The GA selects one or more members of the population based on some measure of the member's fitness to act as "parents" for the next generation;
- 2) *Crossover*: The GA combines the selected parents into one or more offspring by combining their genetic material;
- 3) *Mutation*: The genes of the offspring's "chromosomes" are randomly mutated at some given probability into one of the possible values; and

4) *Replacement*: The offspring are added to the population, which may trigger the removal of members judged to be the least “fit” when the population size reaches some upper limit, say N .

To facilitate a genetically modeled recombination of the selected parents and the mutation of their offspring, GA’s use a “chromosome”-like encoding of population members, typically as bit strings with each bit representing a “gene.” Each gene is capable of representing some number of possible values, called “alleles.” For example, in a traditional CS each position in a bit string represents a gene capable of two possible allele value, zero or one. In XCS, the “chromosome” of an individual classifier is composed of a single bit string containing each of the classifier’s conditions and actions appended together in fixed positions.

Selection, the first step in the formation of a new “generation,” relies on some measure of fitness applied to each member of the population in order to direct the GA’s search. The fitness measure of each classifier is typically used to probabilistically select classifiers to act as parents for the next generation.

The process of crossover combines the chromosomes of the parent classifiers into one or more offspring classifiers. The crossover scheme employed in the work presented in this paper combines the two parent’s bit-strings together by probabilistically selecting a parent from which to copy the allele at a given location into the offspring’s chromosome.

During mutation, a random search of the solution space is done to introduce new “genetic material” into the population. This is how the mutation process determines a random alteration of genes within an offspring’s chromosome to one of their possible allele values.

The GA models “survival of the fittest” through replacement, or deletion, of classifiers. The replacement process determines which, if any, classifier to remove from the system in order to make room for a new classifier to be added to the population. Replacement is generally driven by a probabilistic selection function based on the fitness of each population member.

Unlike traditional CS, the GA in XCS operates on a niche of classifiers that share some functional relation [4]. Though the GA in Wilson’s original XCS operated on the set of classifiers matched by the current system input (*i.e.*, the match-set), subsequent work by Wilson and others indicated that operating the GA in the action-set yielded improved performance [5,11].

3. Modifying XCS

This section describes the four modifications investigated by the authors intended to allow XCS to operate more effectively in systems with smaller population sizes. These changes, though relatively minor in scope, provide a drastic improvement in performance in size-constrained systems in both single and multi-step environments.

3.1 Modification 1: Use an experience-based GA trigger function

The first proposed modification to Wilson's original XCS definition [3] was made to the trigger function for the genetic algorithm. Wilson's original trigger algorithm attempted to evenly distribute GA activations across all encountered match sets regardless of their input frequency. This GA was later changed to operate on the action set as this was found to provide better performance [5,11]. Evenly distributing GA activations was achieved by triggering the GA when the average amount of time (*i.e.*, exploration runs) since the last GA activation in the given set of classifiers exceeded a threshold value, θ_{ga} .

While Wilson's trigger algorithm does ensure that GA activations are distributed relatively evenly across the encountered match states (assuming the match states are approximately evenly encountered) it can introduce instability in small populations, as GA activation is then independent of the amount of experience attained by the classifiers in the triggering set. Furthermore, due to the averaging nature of the algorithm, general rules that appear in more action sets and typically have a higher numerosity tend to dominate the original trigger function. This can result in classifiers with high specificity

being largely ignored in the trigger calculation due to their typically lower numerosity and rarity of activation.

While the impact of the original triggering algorithm is generally benign in systems with large population sizes relative to the payoff landscape, the experience-independent nature of the algorithm becomes detrimental in size-constrained systems, where missteps become more costly, unless the value of θ_{ga} is increased significantly. However, this increase results in a diminished learning rate, as a large portion of the population may be satisfactorily evaluated but unable to be operated upon by the GA.

The authors propose the use of a new experience-based trigger algorithm to address the aforementioned shortcomings. Under this alternate algorithm, the GA is only triggered when the activated action set contains any number of classifiers, with experience since the last GA trigger on that set, greater than the trigger threshold (*i.e.*, when $exp_{cl} > \theta_{ga}$). This experience-based triggering ensures that at least a portion of the set of classifiers being operated upon by the GA has been sufficiently evaluated and that specific and rarely activated classifiers can trigger the GA without being overwhelmed by generalists in the same action set.

3.2 Modification 2: Use max-deletion in the deletion-selection algorithm

The second change proposed to Wilson's original XCS implementation [3] was made to the deletion-selection algorithm. In XCS, a classifier is removed from the population whenever the population size exceeds the specified maximum (N). In order to remove a classifier, the system must make some judgment as to which classifiers are more or less valuable to the system. In his work, Wilson presented two deletion-vote

functions – we focus on the second type here, as it appears to offer greater performance in constrained systems.

Wilson’s deletion algorithm operates by assigning each classifier a deletion “vote” which is the product of the classifier’s numerosity (*num*) and the learned action set size estimate (*as*). The classifier’s numerosity (*num*) indicates the number of identical classifiers in the population represented by the given classifier, and the learned action set size estimate (*as*) represents an estimate of the number of classifiers generally present in the action sets to which the classifier belongs. Furthermore, if the classifier’s experience is greater than the deletion threshold (*i.e.*, when $exp_{cl} > \theta_{del}$) and it’s fitness is less than a constant fraction (δ) of the mean fitness of the population, then the deletion vote is scaled by the difference between the classifier’s fitness and the mean population fitness. That is, Wilson’s deletion algorithm calculates votes as follows:

$$1) \text{ vote}_{cl} = num_{cl} * as_{cl}$$

$$2) \text{ IF } ((exp_{cl} > \theta_{del}) \text{ AND } (f_{cl} / num_{cl} < \delta * (\sum f_{pop} / \sum num_{pop})))$$

$$\text{ vote}_{cl} = \text{ vote}_{cl} * (\sum f_{pop} / \sum num_{pop}) / (f_{cl} / num_{cl}).$$

Once the deletion vote of each classifier is determined, the deletion-selection algorithm selects a classifier for removal. Wilson’s deletion-selection algorithm uses *roulette selection* where a classifier is probabilistically selected for deletion from the population based on its deletion vote. While roulette deletion-selection is adequate in environments with a large population size, in more size-constrained environments the use of roulette deletion-selection can negatively affect system performance as its probabilistic nature often results in the deletion of effective and useful classifiers.

The authors propose the use of *max-deletion* in the deletion-selection algorithm to allow XCS to operate more effectively in size-constrained environments. Under *max-deletion*, the system simply removes the classifier with the highest deletion vote, or chooses one randomly from the set of classifiers with the highest vote should there be more than one. Because of using max-deletion, the system maintains an implicit downward pressure on the numerosity of classifiers, which is doubly useful for the following two reasons.

First, it prevents the favoritism inherent in the proposed experience-based GA trigger algorithm from overly saturating the population with frequently matched classifiers, as the GA will tend to result in an increased numerosity and action set size estimate of such classifiers. Second, this downward pressure on classifier numerosity allows the smaller population of a size-constrained system to maintain a more diverse population of classifiers.

3.3 Modification 3: Intensify the GA parent-selection process

The third modification proposed is to intensify the GA parent-selection process. In a traditional XCS, the GA probabilistically selects the parents for the generation of new offspring from the current action-set based on the value of each classifier's fitness [3]. For size-constrained systems, the selection probability needs to be intensified by using the fitness of each classifier taken to the power, i . By intensifying the difference between high and low fitness classifiers in the selection set, a size-constrained system is less likely to select low fitness classifiers and therefore focuses the GA on fitter members of the population.

While reducing the probability that lower fitness classifiers in a set will be selected for reproduction clearly narrows the search provided by the GA, it also becomes beneficial in small populations as the likelihood of wasting population space on inferior classifiers is reduced.

3.4 Modification 4: Update parameters on both exploration and exploitation episodes

In an effort to avoid favoring classifiers frequently chosen for activation, XCS limited the updates of classifier parameters such as experience, error, fitness and predicted payoff, during exploration steps only. This tends to evenly distribute parameter updates over the population due to the random nature of exploration steps. The fourth and final modification proposed by the authors on the timing of parameter updates performed by the system is simply to update classifier parameters (i.e., error, prediction and action set size estimate) with the exception of a classifier's experience and fitness, on both exploration and exploitation steps. The reasoning behind this change is two fold.

First, the information gained from exploitation steps is typically lost in XCS. This information is valuable as long as it does not negatively bias the system towards classifiers chosen on exploitation, particularly so in a size-constrained system where the limited population size reduces the learning rate of the system. Second, by not incrementing the classifier experience count on exploitation steps, the GA is not biased towards those classifiers typically chosen for their high payoff prediction.

4. Experimental Results

This section presents experimental results supporting the use of the proposed modifications to XCS for size-constrained systems. Results from the first experiment, the 6-Multiplexer problem [3], demonstrate the modified system's superior performance in a single step problem with two payoff levels. Results from the second experiment, the Woods-2 problem [3], further demonstrate the improvement in performance gained with the proposed modifications in a multi-step problem with delayed rewards.

All data gathered for this paper used a version of Barry's JXCS classifier system API [13] modified by the authors to follow the implementation described in [7]. All system parameters are as in [7] except where noted. Covering was set to generate 1 classifier per match-set (i.e. $\theta_{mna}=1$), as this yields better performance in size-constrained systems. Subsumption deletion was used in all experiments, though the experience threshold was set to twice the GA threshold (i.e., $\theta_{sub}=2*\theta_{ga}$). This was found to reduce erroneous subsumptions caused by the occasional situation of an overly general classifier encountering only those match states that it correctly maps to actions, thereby momentarily appearing to be accurate enough to subsume other more truly accurate classifiers. The higher threshold provides an occasionally inaccurate classifier more opportunities for deletion before being afforded the chance to subsume other classifiers.

4.1 Experiment 1: The 6-Multiplexer Problem

The 6-Multiplexer (6-Mux) problem is commonly used in the literature as a benchmark experiment to test and compare classifier systems. The 6-Mux problem is a

single step classification task in which the system must learn to map a 6-bit input string to one output bit. Based on the logic circuit of the same name, the multiplexer operates by mapping two of the six input bits (*i.e.*, the address bits) to one of the remaining four input bits and placing the mapped bit's value on the output bit.

The multiplexer class of problems are frequently used in the literature as they present a challenging single step task with a solution space capable of testing the systems ability to develop accurate, general and optimal rules. Only three of the bits in any given input string of a 6-Mux system are important, the two address bits and the bit at the position encoded by the address bits. Due to this property the system can develop general rules with don't-care symbols in the ignored bit positions, which will accurately encode the entire set of classifiers, masked by that bit string. For example, while the 6-Mux problem has 64 possible inputs, a smaller set of 16 optimal classifiers can accurately cover the entire input space given a two-level payoff landscape (*i.e.*, rewards of 0 and 1000 for incorrect and correct outputs, respectively).

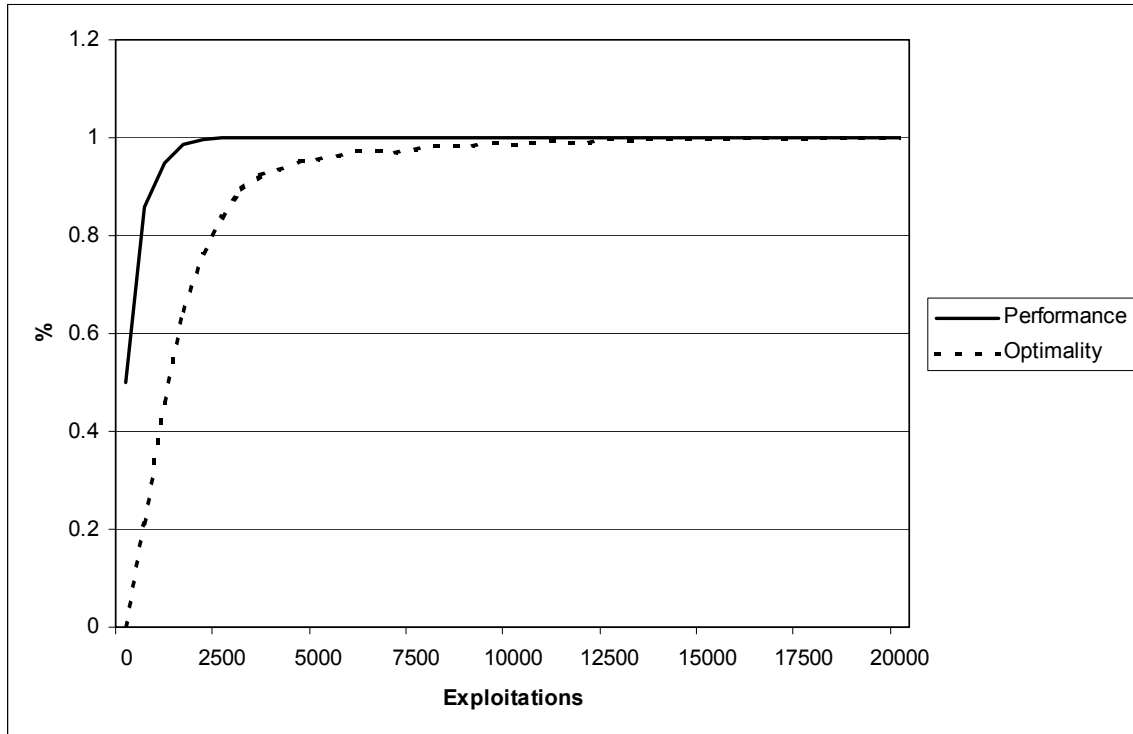


Figure 1. Performance and optimality of standard XCS in 6-Mux with $N=400$, $\theta_{ga}=25$, $\theta_{sub}=50$ (average of 100 runs).

Figure 1 shows the performance and optimality (*i.e.*, percentage of the optimal set of 16 classifiers found in the current population) achieved by a standard XCS with the standard population size of $N=400$. As shown, the system is able to quickly learn a high performance population of classifiers which includes the optimal of classifiers. These results approximate those reported by Wilson [3] and Kovacs[5]. There are two factors that cause the slight difference in optimality rates. (Note: Kovacs reported 100% optimality by the 8000th iteration.) The first is the use of a different deletion-selection function than was used in Kovacs, as that function was found to be detrimental in small populations. The second is the reduced covering operator.

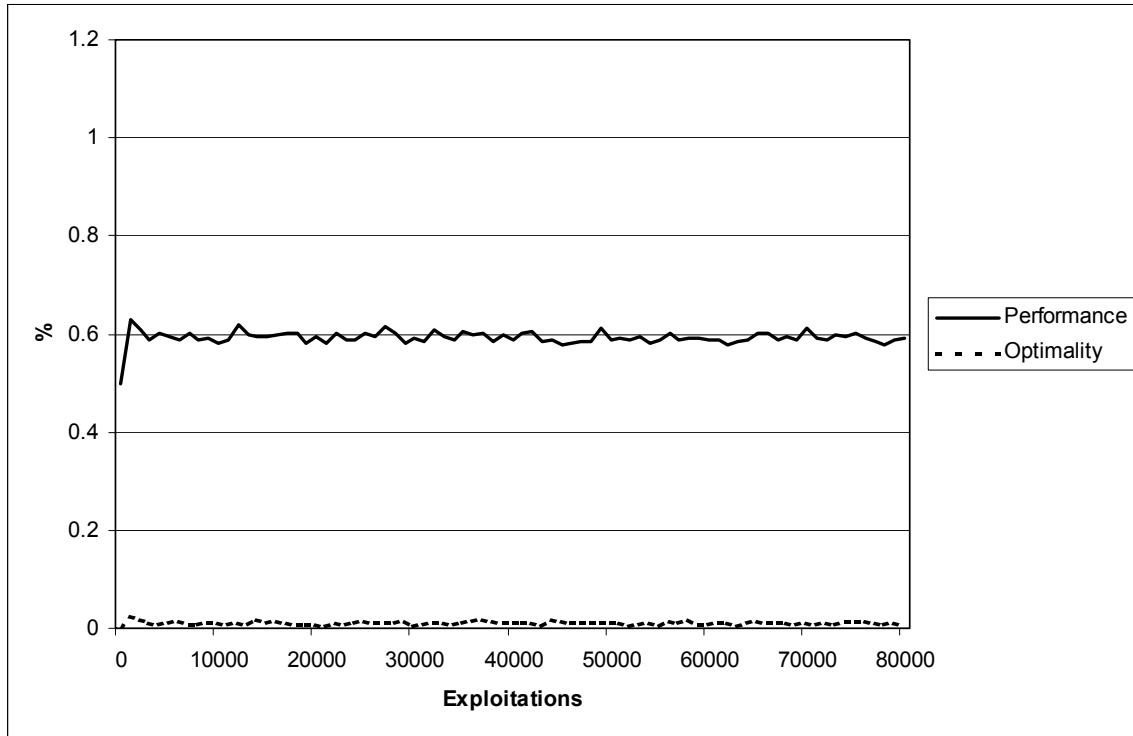


Figure 2. Performance and optimality of standard XCS in 6-Mux with $N=40$, $\theta_{ga}=25$, $\theta_{sub}=50$ (average of 100 runs).

Figures 2 and 3 show the performance of the original XCS and the modified implementation; respectively, in a size-constrained 6-Mux environment with a maximum population size of $N=40$ classifiers. As can be seen in Figure 2, the standard XCS is unable to achieve a stable population while the modified version shown in Figure 3 achieves both high performance and an optimal population. The number of iterations required for the modified system to achieve the optimal solution is higher because of the small population size slowing the evolutionary search. It should be noted that the modified system is capable of a much higher learning rate but the attempt was made to retain as much similarity between the two systems for the sake of comparison.

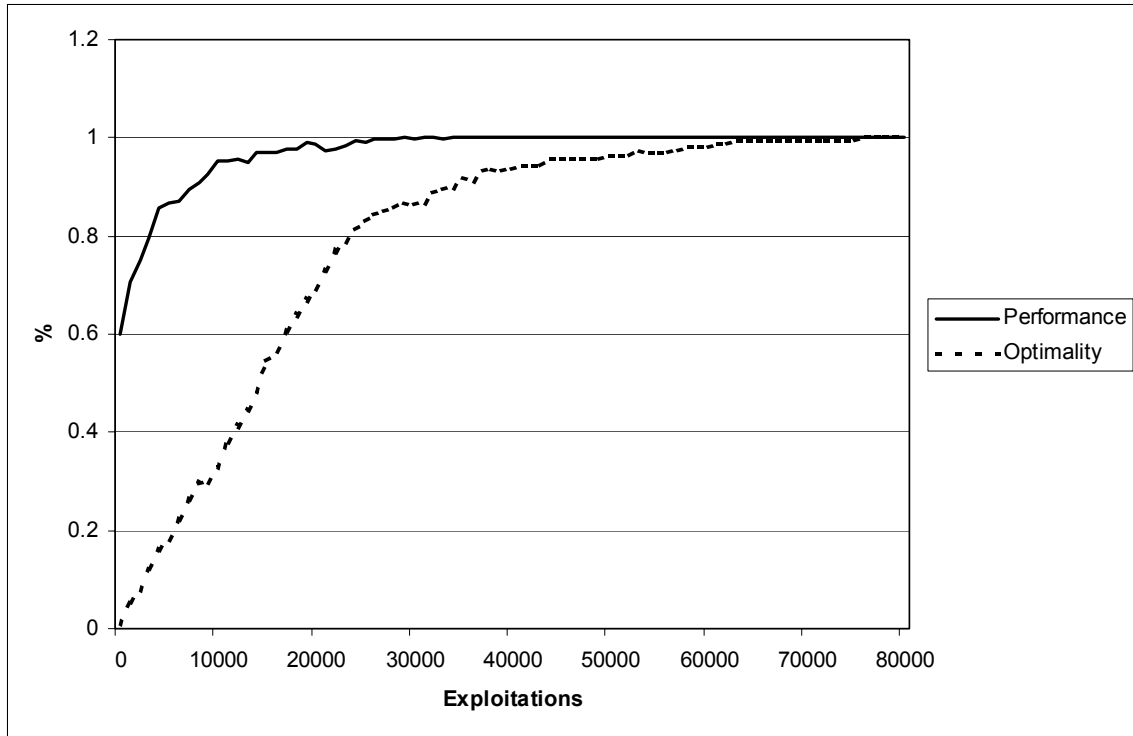


Figure 3. Performance and optimality of modified XCS in 6-Mux with $N=40$, $\theta_{ga}=25$, $\theta_{sub}=50$, $i=10$ (average of 10 runs).

4.2 Experiment 2: The Woods-2 Problem

This section compares the performance of XCS and the modified XCS in the Woods-2 problem [3]. Woods-2 is a multi-step problem with delayed rewards in which the system represents an “animat” [14], or organism, searching for food on a 30 by 15 grid. The grid is composed of five possible location types (encoded as three bit binary strings): empty space through which the animat may move, two types of “rock” which block movement, and two types of food which garner a reward when occupied by the animat. The grid contains a repeating pattern of 3 by 3 blocks with food in the upper right block

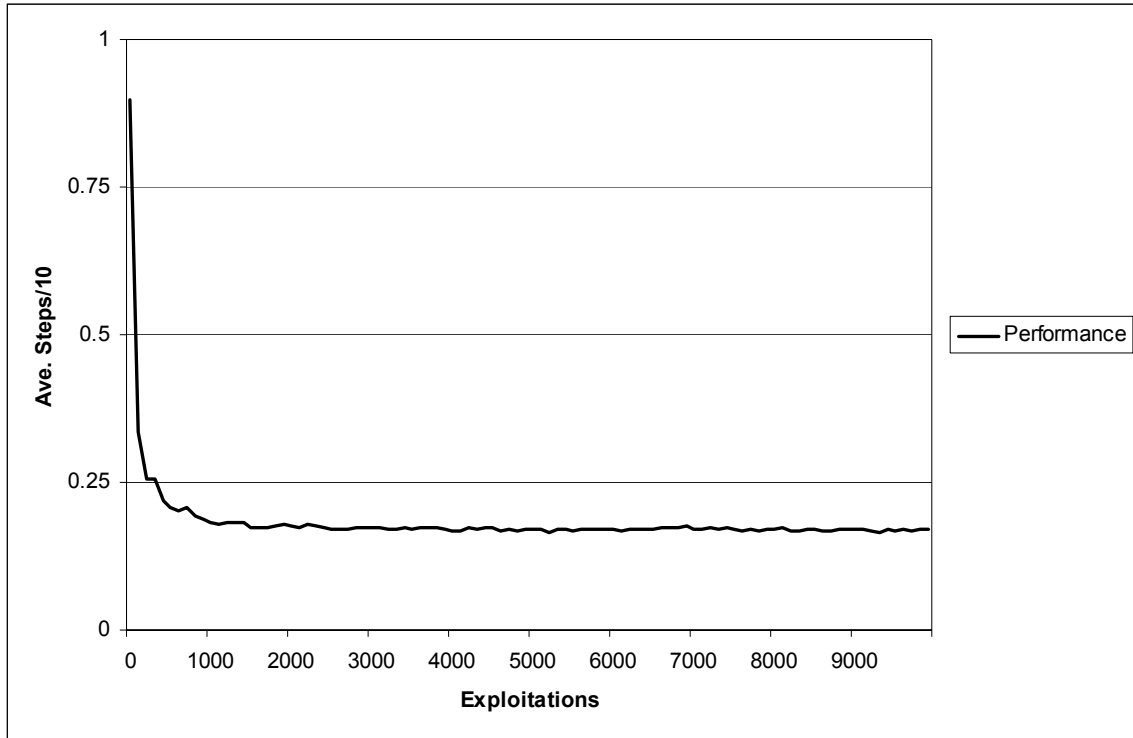


Figure 4. Performance of XCS in Woods-2 with $N=800$, $\theta_{ga}=25$, $\theta_{sub}=50$ (average of 10 runs).

position, separated by empty space and wraps around (*i.e.*, moving off the right edge of the grid places the animat on the left edge).

The system input is a 24 bit binary string representing the eight space types surrounding the location currently occupied by the animat (*i.e.*, the animats “sensed” surroundings). The output of the system is a three bit binary string representing in which of the eight possible directions the animat attempts to move. At the start of each episode, the types of food and rock in each block are randomly selected but the locations remain constant. This is done to better test the generalization abilities of the system.

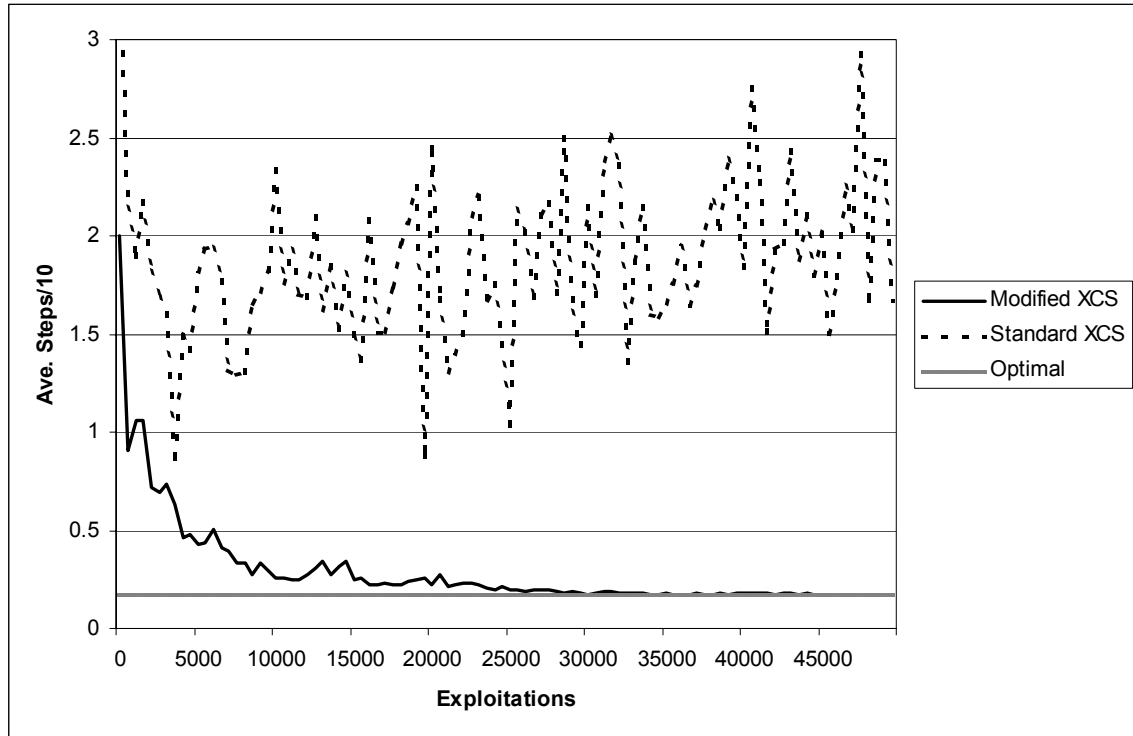


Figure 5. Performance of standard XCS and modified XCS in Woods-2 with $N=80$, $\theta_{ga}=50$, $\theta_{sub}=100$, $i=10$ (average of 10 runs).

Figure 4 shows the performance (measured in average number of steps required for the animat to reach food divided by 10) of a standard XCS in the Woods-2 problem with a population size of $N=800$. These results approximate those reported by Wilson [3] and demonstrate the ideal solution of 1.7 steps on average anticipated for the problem.

Figure 5 shows the performance of both standard XCS and the modified XCS in the Woods-2 environment with a population size of $N=80$ (which again is one order of magnitude smaller than was used in [3]). As shown, the modified system is able to achieve a high level of performance while the standard system is unable to develop a stable population of classifiers.

5. Conclusions and Recommendations

This paper has presented four modifications that allow XCS to operate with higher performance in size-constrained systems. Experimental results presented demonstrate that these modifications allow the system to make greater use of the available population resources in a system an order of magnitude smaller than is typically used in the literature.

While it is possible to achieve a compact and optimal solution with an unmodified XCS given a large population size, the ability to achieve similar performance in a size-constrained system is likely to be of value in a variety of more complicated problems where system resources are limited relative to the size of the payoff landscape. Furthermore, due to the presence of population-wide functions in the XCS implementation (*e.g.*, deletion-selection calculations), increases in population size result in a significant increase in time required to reach a compact solution. Therefore, depending on the complexity of the problem being solved and the amount of generalization possible, it is possible that the tradeoff between population size and time-to-solution will favor the use of the modified system.

The authors are optimistic that the work presented in this paper will perhaps be applicable in software systems for bioinformatics research [15]. For example, some real-time data mining systems can be organized so that fresh data from wet labs can be kept in a sort of short-term memory while the rest of the (massive amounts of) data is maintained in more long-term memory. The modified XCS proposed in this paper can be applied to data considered “short-term.” Findings can be integrated with classification information

derived from the larger data set. Wilson has already started work on XCS applications in data mining [16] and function approximation [17].

6. References

- [1] Holland J.H.: Adaptation. In: R. Rosen, F.M. Snell (Eds.), *Progress in Theoretical Biology*, 4, 1976, Plenum Press, NY.
- [2] Holland J.H.: Adaptive algorithms for discovering and using general patterns in growing knowledge bases. *International Journal of Policy Analysis and Information Systems*, 4(3), 1980, 245-268.
- [3] Wilson S.W.: Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2), 1995, 149-175.
- [4] Booker L.B.: *Intelligent behavior as an adaptation to the task environment*, Ph.D. Dissertation, 1982, Computer and Communication Sciences, The University of Michigan.
- [5] Kovacs T.: *Evolving Optimal Populations with XCS Classifier Systems*. Master's thesis, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996.
- [6] Wilson S.W.: ZCS: A zeroeth level classifier system, *Evolutionary Computation*, 2(1), 1994, 1-18.
- [7] Butz M.V., Wilson S.W.: *An Algorithmic Description of XCS*, IlliGAL Report No 2000017, April 2000, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.
- [8] Watkins C.: *Learning from Delayed Rewards*. Ph.D. Dissertation, 1989, Cambridge University.
- [9] Watkins C., Dayan, P.: Technical note: Q-learning. *Machine Learning*, 8, 279-292, 1992.
- [10] Holland J. H.: *Adaptation in natural and artificial systems*, 1975, Ann Arbor, University of Michigan Press.

- [11] Wilson S.W.: Generalization in the XCS classifier system. In: J.R. Koza *et al.* (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 665-674, 1996, Morgan Kaufmann, San Francisco, CA.
- [12] Wilson S.W.: Explore/exploit strategies in autonomy. In: P. Maes *et al.* (Eds.), *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pp. 325-332, 1996, The MIT Press, Cambridge, MA.
- [13] Barry A.: JXCS, 1999. <http://www.cs.bath.ac.uk/~amb/LCSWEB/jxcsawt.zip> (current July 23, 2002).
- [14] Wilson S.W.: Knowledge growth in an artificial animal. *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, (pp. 16-23), 1985, Hillsdale, New Jersey: Lawrence Erlbaum Associates.
- [15] Heath L.S., Ramakrishnan N.: The emerging landscape of bioinformatics software systems. *IEEE Computer*, 35(7), 2002, 41-45.
- [16] Wilson S.W.: Mining oblique data with XCS. In: P.L. Lanzi *et al.* (Eds.), *Advances in Learning Classifier Systems (Lecture Notes in Computer Science, Vol. 1996)*, pp. 158-176, 2001, Springer-Verlag, Heidelberg, Germany.
- [17] Wilson S.W.: *Classifiers that approximate functions*. IlliGAL Report No 2001027, September 2001, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.