

IMPROVING EXTENDED CLASSIFIER SYSTEM PERFORMANCE  
IN RESOURCE-CONSTRAINED CONFIGURATIONS

---

A Thesis  
Presented  
to the faculty of  
California State University, Chico

---

In Partial Fulfillment  
of the Requirements for the Degree  
Master of Science  
In  
Computer Science

---

by  
© Devon L. Dawson 2002

Fall 2002

IMPROVING EXTENDED CLASSIFIER SYSTEM PERFORMANCE  
IN RESOURCE-CONSTRAINED CONFIGURATIONS

A Thesis

by

Devon Dawson

Fall 2002

APPROVED BY THE DEAN OF THE SCHOOL OF GRADUATE,  
INTERNATIONAL, AND SPONSORED PROGRAMS:

---

Robert M. Jackson, Ph.D.

APPROVED BY THE THESIS ADVISORY COMMITTEE:

---

Benjoe Juliano, Ph.D., Chair

---

Anne M Keuneke, Ph.D.

## PUBLICATION RIGHTS

No portion of this thesis may be reprinted or reproduced in any manner unacceptable to the usual copyright restrictions without the written permission of the author.

## TABLE OF CONTENTS

	PAGE
Publication Rights.....	iii
List of Figures.....	vi
List of Symbols.....	viii
Abstract.....	xi
 CHAPTER	
I. Introduction.....	1
II. Classifier Systems: Strength Based Fitness.....	5
Classifier System Overview.....	6
Classifier System Components.....	8
Difficulties Encountered in the Use of Strength-Based Fitness.....	14
III. Extended Classifier Systems: Accuracy Based Fitness.....	18
Extended Classifier Systems Overview.....	19
Generality and Optimality of Classifiers.....	28
IV. Improving Extended Classifier System Performance.....	30
Proposed Modifications for Accuracy-Based Systems.....	31
Experimental Evaluation of Proposed Modifications.....	38
Summary of Results.....	51
V. An Effectiveness-Based Extended Classifier System.....	52
Modifications Required for Effectiveness-Based Extended Classifier Systems.....	54

CHAPTER	PAGE
Experimental Comparison of Accuracy-Based and Effectiveness-Based Systems.....	60
Summary of Results.....	68
VI. Conclusions And Future Research.....	70
Conclusions.....	70
Future Research .....	72
References.....	77
Appendices	
A. Tables.....	81

## LIST OF FIGURES

FIGURE	PAGE
1. A Typical Classifier System .....	7
2. Performance of XCS and MXCS in 6-Mux with N=400 and System Configurations as in Table A-1 .....	41
3. Optimality of XCS and MXCS in 6-Mux with N=400 and System Configurations are as in Table A-1 .....	42
4. Performance of XCS and MXCS in 6-Mux with N=40 and System Configurations as in Table A-1 .....	44
5. Optimality of XCS and MXCS in 6-Mux with N=40 and System Configurations as in Table A-1 .....	45
6. The Woods-2 Environment with Food Represented by "F" and "G", Rock Represented by "O" and "Q", Empty Space Represented by "." and the Animat Represented by "*" .....	47
7. Performance of XCS and MXCS in Woods-2 with N=800 and System Parameters as in Table A-2 .....	48
8. Performance of XCS and MXCS in Woods-2 with N=80 and System Parameters as in Table A-2 .....	50
9. Performance of EXCS and MXCS in 6-mux with N=40 and System Parameters as in Table A-3 .....	62
10. Optimality of EXCS and MXCS in 6-mux with N=40 and System Parameters as in Table A-3 .....	63
11. Performance of EXCS and MXCS in 6-mux with N=20 and System Parameters as in Table A-3 .....	64
12. Optimality of EXCS and MXCS in 6-mux with N=20 and System Parameters as in Table A-3 .....	65

FIGURE	PAGE
13. Performance of EXCS and MXCS in Woods-2 with N=80 and System Parameters as in Table A-4 .....	68
14. Performance of EXCS and MXCS in Woods-2 with N=40 and System Parameters as in Table A-4 .....	69

## LIST OF SYMBOLS

### SYMBOL

- [P] : The population, a set containing all classifiers in the system.
- [M] : The match set, contains all classifiers whose conditions are satisfied by the current input state on any given iteration.
- [A] : The action set, a subset of any [M] containing all classifiers advocating the same action.
- [A]<sub>-1</sub> : The prior action set, the action set selected by the conflict resolution process for activation on the previous iteration if the end of the problem was not reached.
- $N$  : The maximum population size.
- $S$  : The size of the problem search space.
- $P$  : The payoff received from the environment on a given iteration.
- $P_{-1}$  : The payoff received on the previous iteration if the end of the problem was not reached.
- $\beta$  : The learning rate ( $0 < \beta \leq 1$ ), used in updating classifier parameters  $p_{cl}$ ,  $\varepsilon_{cl}$ ,  $as_{cl}$ ,  $f_{cl}$ .
- $\gamma$  : The Q-Learning discount factor used in multi-step environments ( $0 < \gamma \leq 1$ ).
- $p_{cl}$  : The payoff predicted by an individual classifier.
- $f_{cl}$  : The fitness of an individual classifier.

- $\varepsilon_{cl}$  : The learned error in prediction,  $p_{cl}$  (*i.e.*, the difference between  $p_{cl}$  and  $P$ ) for a classifier.

- $\varepsilon_0$  : The accuracy criterion, the minimum prediction error below which a classifier is defined to be completely accurate.

- $k_{cl}$  : The accuracy of a classifier as a function of its error,  $\varepsilon_{cl}$ .

- $\alpha$  : The accuracy function fall-off rate.

- $v$  : The accuracy function exponent.

- $as_{cl}$  : The learned average size of the action sets in which the classifier is a member.

- $exp_{cl}$  : The experience of a classifier counting the number of times its action has been activated on exploration steps.

- $num_{cl}$  : The numerosity of a classifier – counts the number of microclassifiers represented by the given macroclassifier.

- $\theta_{del}$  : The deletion experience threshold.

- $\theta_{ga}$  : The GA trigger threshold.

- $\theta_{mna}$  : The minimum number of actions required in any given match set below which covering is triggered.

- $\theta_{sub}$  : The subsumption experience threshold which must be reached by a classifier before it can subsume another more specific classifier.

- $P\#$ : The generality rate, the probability that a gene will be toggled to a don't-care bit in the generation/mutation of classifiers.

- $\mu$ : The mutation probability.

- $\chi$ : The crossover probability.

- $spec_{cl}$ : The specificity of a classifier, equal to the percentage of a classifier's condition bits that aren't don't-cares (#).

- $\delta$ : the mean population fitness fraction below which a classifier's fitness is considered in the deletion-vote function.

- $\theta_{sub[A]}$ : The action set subsumption threshold, the amount of experience required for a classifier to subsume another during action set subsumption.

- $\theta_{cull}$ : The culling threshold, the maximum percentage of the population that may consist of ineffective classifiers.

- $\psi$ : The accuracy discount factor.

- $\Omega$ : The constraint measure for a problem/population size configuration.

- $eff_{cl}$ : The effectiveness of a classifier.

## ABSTRACT

### IMPROVING EXTENDED CLASSIFIER SYSTEM PERFORMANCE IN RESOURCE-CONSTRAINED CONFIGURATIONS

by

© Devon L. Dawson 2002

Master of Science in Computer Science

California State University, Chico

Fall 2002

An extended classifier system, or XCS, is a rule-based machine learning technique that uses an evolutionary search and reinforcement learning technique to develop a population of rules capable of complete and accurate mappings of a problem's payoff landscape. The resources required, however, for XCS to develop a complete map using the standard implementation can be extensive and therefore potentially problematic as it hinders the ability of XCS to be applied to larger and more complex problems without requiring a prohibitive increase in system resources. In an effort to better understand and improve upon the resource requirements of XCS this work investigates the limiting case provided by applying XCS to size-constrained systems, that is, where the size of the population is significantly smaller than the size of the search space.

This work begins with an introduction to traditional strength-based classifier systems and the subsequently developed accuracy-based XCS. The first set of modifications is then presented which consists of minor functional changes made to XCS with the intent of making better use of the limited population resources available. Experimental results are then presented showing the superior performance achieved through the use of the first set of modifications in single-step and multi-step problems in size-constrained systems where the standard XCS implementation is unsuccessful.

The second set of modifications then introduces the effectiveness-based extended classifier system, or EXCS, developed to further improve the performance of an XCS-based system operating in even more highly size-constrained systems. Whereas classifier fitness in XCS is a function of the classifier's prediction accuracy, EXCS uses an effectiveness-based fitness measure combining a classifier's prediction magnitude, prediction accuracy and condition generality. Experimental results are presented demonstrating that EXCS outperforms the modified XCS in single-step and multi-step problems when the population size is severely limited and the problem allows the use of an incomplete map.

## CHAPTER I

### INTRODUCTION

Classifier systems (CS) are a rule-based machine learning paradigm first introduced by Holland [1, 2] as a framework with which to study the application of his genetic algorithms (GA) [3] to learning in rule-based systems with sparse or delayed rewards. CS have traditionally relied on a strength parameter to both determine the best action to take in a given situation and to measure classifier fitness. However, the use of strength-based fitness has since been found to be a potentially misleading indicator of a classifier's value to the system [4, 5, 6, 7] and hence a poor metric to use in guiding the action selection and classifier discovery aspects of a CS (*i.e.*, the GA).

The extended classifier system, or XCS, developed by Wilson [4], uses the accuracy of classifier payoff predictions as the sole measure of a classifier's fitness. By using an accuracy-based fitness function as well as a niched genetic algorithm [8] XCS has been shown to maintain an implicit pressure towards developing a complete, accurate and compact mapping of  $X \times A \Rightarrow P$ . That is, a mapping from system state (*i.e.*, inputs) to classifier actions (*i.e.*, outputs) to anticipated environmental payoff (*i.e.*, reward) [4]. Furthermore, XCS appears to possess an inherent tendency to work towards the development of maximally general and optimal classifiers [4, 6, 9] whose conditions cover the maximum number of system inputs while still remaining accurate. This ability to generalize allows XCS to develop compact representations of payoff landscapes by

taking advantage of any overlap in the input:output:reward search space.

While XCS has been shown to be capable of learning complete, accurate and maximally general mappings of a payoff landscape, in many single-step and multi-step environments the resources required to do so can be extensive. In an effort to better understand and improve upon these requirements, this work investigates the application of XCS-based systems to size-constrained population/problem configurations. For the purposes of this work we define a *size-constrained* system to be one where the population size,  $N$ , is less than the size of a complete input:output:reward map,  $S$ . It was hoped that such an investigation would provide some insight into the internal XCS mechanisms that lead to the resource requirements exhibited in the literature and prompt the development of modifications that might alleviate these requirements to some degree.

After an introduction to traditional strength-based classifier systems and the subsequently developed accuracy-based XCS, this work presents two sets of modifications aimed at addressing the resource requirements of XCS. The first set of changes, presented in Chapter IV, are intended to allow a slightly modified XCS to achieve much higher performance in size-constrained systems while retaining XCS' ability to develop a complete payoff map. Experimental results presented at the end of Chapter IV support the use of the proposed modifications in size-constrained single-step and multi-step environments by demonstrating the improved performance observed when using the modifications.

Although the results presented for the first set of modifications show the ability of XCS (both modified and unmodified) to develop a complete map this same completeness can be seen to be problematic in some practical real-world problems where the payoff landscape may contain a large number of very low payoff states and or rarely encountered conditions relative to the available system resources. As XCS considers a rule that accurately predicts that no reward will be gained through its activation to be as fit and valuable to the system as one which accurately predicts the return of the maximum reward possible, XCS will devote significant resources in the form of time (*i.e.*, computing cycles) and storage (*i.e.*, the set of classifiers in the population) to ineffective but accurate classifiers in the process of developing a complete map.

In addressing this issue Wilson [4] states,

Adherents of payoff-based fitness might suggest that the efficiency issue arises because accuracy-based fitness, as demonstrated, results in relatively complete maps of the payoff landscape, whereas traditional classifier systems ‘go for the best’ (-paying classifiers) and ignore the rest. They might say that the latter pragmatic approach is the only practical one in large problems (Holland et al., 1986). Against this one can note that the traditional classifier system has no principled approach to achieving generalization – the lack of which may well offset whatever is gained through pragmatics – and the solutions converged upon are often suboptimal. Nevertheless, in many problems large regions of the  $X \times A \Rightarrow P$  map will be relatively unremunerative, and techniques for reducing exploration there need to be developed.

In an effort to address the issue of pragmatism identified by Wilson, Chapter V presents an effectiveness-based fitness measure and the second set of modifications, which appear to be necessary in order to implement the effectiveness-based XCS (EXCS). Though these modifications do not appear to benefit an accuracy-based system like XCS (or the modified XCS presented in Chapter IV), they allow the effectiveness

based EXCS to make better use of population resources by developing an incomplete, and therefore smaller map of the payoff landscape, focused on the more effective classifiers for each input condition. Though EXCS develops an incomplete map, it is shown to retain XCS' ability to develop maximally general and optimal rules.

Experimental evidence provided in Chapter VI shows that EXCS markedly outperforms the modified XCS presented in Chapter IV when run in single-step and multi-step environments with severe size constraints.

## CHAPTER II

### CLASSIFIER SYSTEMS:

#### STRENGTH-BASED

#### FITNESS

Classifier Systems (CS), first introduced and subsequently refined by Holland [1, 2], are a class of machine learning techniques which use a biologically inspired process of natural selection and genetic adaptation [3, 10] to direct an adaptive search in a rule-based system. Building upon work first introduced by Holland [3] developed CS as a framework with which to test the application of evolutionary operations, such as fitness-based selection, genetic crossover and mutation, to learning and adaptation in an environment with sparse or delayed rewards (*i.e.*, where the reward returned by the environment may occur some number of steps after the action garnering that reward is taken).

This chapter describes the architecture of a standard CS and discusses the use of genetic algorithms [3, 10] to search for a solution in a problem domain. Though an alternate implementation of CS was subsequently developed in which the GA operates on entire populations of classifiers (*i.e.*, the Pittsburgh, or *Pitt.* approach [11]) the majority of CS research has focused on the system outlined by Holland (*i.e.*, the Michigan, or *Mich.*, approach [1, 2]) in which a single population of classifiers is optimized with the

GA operating on individual classifiers within a single population. This work focuses exclusively upon the latter Mich. approach.

Note that while Holland's original CS differed slightly from that given here, the fundamental descriptive and functional components remain the same - certain simplifications are made for brevity and in order to provide a common foundation applicable to both strength-based CS and the subsequently developed accuracy-based XCS [4].

### Classifier System Overview

As shown in Figure 1, Holland's classifier system consists of seven main components / processes:

1. A Message list
2. A population of classifiers
3. Environment detectors (*i.e.*, inputs)
4. Environment effectors (*i.e.*, outputs)
5. A conflict resolution component
6. A credit distribution component
7. A genetic algorithm component

The execution of single *iteration* in a CS typically proceeds as follows:

1. Inputs are sensed by the detectors and placed on the message list to encode the currently sensed environmental state.



6. The credit distribution component transfer's strength from the activated rules to the producers of the messages that they consumed (*i.e.*, the message(s) that matched the fired classifier's conditions).
7. Any external reward received is distributed to the classifier(s) active on the last clock cycle.
8. When certain threshold criteria are met the genetic algorithm is invoked and performs the process of fitness-based selection, crossover, mutation and replacement of classifiers.

### Classifier System Components

#### Messages

*Messages* in a CS are traditionally composed of fixed-length bit strings defined over the binary alphabet  $\{0,1\}$ . Messages can originate from one of two sources, external inputs and activated classifiers. Environmental inputs conceptually generate messages through the *detectors* (*i.e.*, system inputs) – which in practice is often merely the act of placing inputs on the message list. Individual classifiers generate messages when activated, or *fired*, which may be used by other classifiers on the next clock cycle or may trigger the *effectors* (*i.e.*, system outputs) to perform some external action.

#### Message List

The active behavior of a CS is the result of the interaction of classifiers with the external environment and each other via the message list. The *message list* is simply the set of messages generated either by classifiers within the system on the previous cycle or by external system inputs sensed by the detectors as the current environmental state.

## Classifiers

*Classifiers* in a CS are simple condition:action rules. The condition part of a classifier consists of one or more bit string masks (to be specific, two were used in the original work by Holland while XCS uses a single condition per classifier). Differing slightly from the binary alphabet of messages, conditions are defined over the trinary alphabet  $\{0,1,\#\}$  - where  $\#$  implies a ‘don’t-care’ bit in the mask. A *don’t-care* symbol indicates a position in the condition that matches any value in the same position of any given message string. While Holland’s original CS definition allowed for an additional *negation* symbol,  $\{-\}$ , which when prefacing a mask (e.g.,  $\{-1,\#\,\#\}$ ) indicated that the mask was a match if the message list contained no message matching the mask, it is common in the XCS literature to simply use the ternary set,  $\{0,1,\#\}$ .

A classifier’s condition is considered to be *satisfied* if one or more messages on the message list match 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  $\{1, 1, 0\}$  existed on the message list. When the condition of a classifier matches one or more of the messages on the message list it is considered a candidate for activation by the conflict resolution component.

The action portion of a classifier is simply another bit string defined over the binary language of  $\{0,1\}$ . If chosen for activation by the conflict resolution component (described below) then the classifier’s action bit string is placed on the message list as a message for the next clock cycle.

### Conflict Resolution Component

The *conflict resolution* component in a CS is responsible for determining which of the candidate rules in the *match set*, [M], (*i.e.*, the set of rules whose conditions were matched by one or more of the messages on the message list) will be selected for activation and hence have their actions placed on the message list for the next clock cycle. The conflict resolution strategy originally proposed by Holland uses an economically derived process of bidding where the *bid* of each rule is determined by a function of that rule's accumulated strength and its specificity as well as a constant,  $k$ . Hence a classifier's bid,  $B$ , is determined by:

$$B = k * strength * specificity \quad (2.1)$$

In a standard strength-based CS the *strength* of a rule is defined to be the amount of utility, or payoff, accumulated by the rule during the execution of the system. A classifier can gain strength in the form of payment of bids from consumers of that classifier's messages or directly from the environment in response to the classifier being active on a clock cycle when a payoff is received. Strength therefore reflects an estimate of the magnitude of the value of the classifier's activation, either directly or as a step in an activation chain that leads to an eventual payoff.

A classifier's *specificity* is equivalent to the number of 1's and 0's in the classifier's conditions. Hence the greater the number of don't-care symbols (*i.e.*, #'s) in its conditions, the lower the specificity of the rule and the greater the number of possible inputs that would match the classifier's conditions.

The determination of which satisfied classifier's are activated is then accomplished through the use of some form of bid-based competition. This competition can take many forms: a probabilistic selection based on each classifier's bid, activation of all classifiers above a certain threshold (*e.g.*, activate all classifiers with bids above the average bid) or simply activating the highest bidder(s).

#### Credit Distribution Component

The *credit distribution* component of a CS is responsible for assigning credit to classifiers in proportion to their individual contribution toward helping the system gain any external payoff received, either on the clock cycle the rule is active or on a subsequent clock cycle as a result of the classifier's prior activation. Holland's original CS implementation relied on a market oriented process known as the bucket-brigade to distribute credit [1, 2].

Under the *bucket-brigade* algorithm, credit is transferred from rules active at the time a payoff is received back through the chain of activation from message consumers to message producers in the form of the bids previously defined.

#### Genetic Algorithm

The *genetic algorithm* (GA) component of a CS performs an adaptive search of the solution space through the use of a process modeled after Darwinian evolution (GA-based machine learning has since been termed *evolutionary computation*). First introduced by Holland [3], GAs implement an evolutionary search via a process of fitness-based natural selection and genetic variation within a population of solution candidates - with the result being a problem independent search technique with an

implicit pressure towards maximizing the fitness of the population members. While a complete discussion of GA theory and research is beyond the scope of this paper (see [3, 10, 12, 13]), this chapter provides an introduction to the basic principles and processes involved in the application of GA to CS.

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

1. During *selection* the GA selects one or more members of the population to act as parents based on some measure of the member's fitness.
2. The *crossover* operation combines the selected parents into one or more offspring.
3. During *mutation*, the genes of the offspring's chromosomes are randomly mutated at some given probability,
4. During the final step, *replacement*, the newly generated offspring are added to the population, which may trigger the fitness-based removal of existing members should the population size reach some upper limit.

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, in classifier systems the population member's *chromosomes* are encoded as strings with each character in the string representing a *gene* in the chromosome. Each gene is capable of representing some number of possible values, called *alleles*.

In a traditional bit-string based CS each position in a bit string represents a gene capable of two possible allele values, zero or one. 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.

During the process of selection the strength of a classifier is used as its fitness measure, where strength is a measure of the magnitude of the payoffs received as a result of the classifier being activated. The fitness measure of each classifier is then used to probabilistically select population members to act as parents for the next generation.

The process of crossover combines the chromosomal encoding of the parent classifiers into one or more offspring classifiers. Though many crossover techniques are possible the "parameterized uniform crossover" [10] scheme employed in this work 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.

Mutation is the random alteration at a given probability of genes within an offspring's chromosome to one of their possible allele values. This process performs a random search of the solution space and introduces new "genetic material" into the population.

Replacement, or deletion of classifiers, represents the "survival of the fittest" nature of a GA. 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. Replacement in a CS population is typically driven by a probabilistic selection function based on the fitness of each population member.

### Difficulties Encountered in the Use of Strength-Based Fitness

As described, strength-based classifier systems allocate their resources (*i.e.*, population space and GA invocations) to classifiers based on the magnitude of each classifier's strength/fitness measures. While strength-based systems have been shown to be able to achieve some degree of success (*e.g.*, Holland and Reitman's CS-1 [14] and Wilson's animat system [15]) it has been found that a system utilizing strength-based fitness suffers from a number of problems which prevent it from being effective in some single-step and many multi-step problems [4, 5, 6, 7].

Wilson [4] and Kovacs [5] both summarize the weaknesses of strength-based systems with Kovacs being the more recent and refined summary. As demonstrated by Cliff and Ross [7], the difficulties encountered in applying strength-based systems to a variety of problems can be attributed to the following (as well as their interaction): strength-based GA selection and the development of overly general classifiers.

Under a strength-based GA selection scheme the probability that a classifier will be selected to act as a parent is proportional to the classifier's strength measure. Therefore the system will clearly tend to favor the classifiers with the highest strength values, regardless of their accuracy. This bias, termed *greedy classifier selection*, can saturate the population with classifiers from the highly rewarding condition:action states. This focus on high strength classifiers at the expense of others leads the system to develop a partial, or *best action*, map [5] biased towards the most highly rewarding (*i.e.*, best) condition:action pairs in the environment with potential gaps in the map if the bias is too great.

The second fundamental difficulty encountered in strength-based systems is their tendency to develop overgeneral classifiers. In a strength-based system an *overgeneral* classifier is a classifier whose conditions are so general as to cause it to be incorrect on some subset of its matched states. Therefore, though the overgeneral classifier is occasionally incorrect it may appear to have high strength due to the strength accumulated during the situations in which it advocated the correct action.

Though some criteria must be met for a system to be prone to developing overgeneral classifiers, these criteria are found in even fairly trivial problems. First, a classifier must be able to act correctly in some states and incorrectly in others with a different reward received for correct and incorrect actions. Second, the reward function must be *biased*, which means that different actions may receive different magnitudes of rewards in different states (*e.g.*, action A receives a reward of 1000 while action B receives a reward of 100).

While the first criterion is trivial, the second can be avoided if the reward function is restricted to two payoff levels. However, designing a system to use an unbiased reward function can be problematic for two reasons. First, it forces the system to allocate resources equally to all correct actions when in fact it may be desirable to favor exploration of some areas of the environment over others. For example, identifying the likelihood that a vehicle is about to crash is possibly more important than identifying that it is about to run out of fuel. Second, for most multi-step environments, while the reward function itself may be unbiased the credit-distribution process typically results in

an inherently biased reward structure as the distribution process will result in varying reward values depending on the stage at which the classifier acts in the activation chain.

As a result, the bias in the GA towards high strength states can negatively impact the ability of the system to develop long chains of classifiers - which are required to achieve an external payoff in many multi-step problems with sparse, or delayed rewards. In such environments numerous stage-setting actions must be taken before the final reward garnering state is reached but the strength bias in GA selection (*i.e.*, greedy classifier selection) makes it difficult for a strength-based system to explore and maintain classifiers in these stage setting states.

The interaction between a strength-based system's tendency to develop overgeneral classifiers and the problem of greedy classifier selection was identified by Cliff and Ross [7] as a serious detriment to system performance in strength-based systems. The GA will tend to select the 'strongest' classifiers in the population, which will tend to be overgenerals if the reward function is strongly biased. Furthermore, the conflict resolution process will tend to select overgenerals owing to their higher strength though more accurate classifiers may exist in the current input state. This interaction was later termed the problem of *strong overgenerals* by Kovacs [5].

One final disadvantage found in using strength-based fitness, and therefore developing a partial map, was exposed by Hartley [16]. Harley demonstrated that should the underlying payoff landscape change during execution, a system employing a partial map must re-discover and re-evaluate all the condition:action pairs discarded due to their previously low strength. However, a system utilizing a complete map need not rediscover

the newly important regions of the map, rather it merely needs to re-learn the appropriate values for the already existing classifiers.

## CHAPTER III

### EXTENDED CLASSIFIER SYSTEMS: ACCURACY-BASED FITNESS

The extended classifier system (XCS) was introduced by Wilson [4] as a potential solution to many of the problems encountered in strength-based CS. Namely, the difficulties encountered in maintaining long chains of classifiers and a strength-based system's tendency to develop overgeneral classifiers [5]. Wilson's development of XCS grew out of his earlier work on a "zeroth-level" classifier system (ZCS) [17], which simplified the original CS framework.

To this end XCS incorporates a variety of techniques: accuracy-based fitness, a niched GA [8] and a credit distribution mechanism modeled on the Reinforcement-Learning technique of Q-Learning [18, 19]. Furthermore, as in ZCS, XCS simplifies the original CS framework by removing the message list and restricting the system to the activation of a single classifier per clock cycle. These developments were aimed at providing a more theoretically grounded system capable of overcoming the shortcomings of traditional strength-based systems while providing a clearer picture of inner workings of the system.

While previous work had utilized varying forms of accuracy as a factor in the fitness calculation (see [4] for a discussion), XCS demonstrated that accuracy alone was an effective and possibly superior fitness measure [4]. Furthermore, by incorporating the

use of a niched GA [8], XCS was shown to be the first classifier system capable of developing complete and accurate mappings of the payoff landscape [4, 9].

### Extended Classifier Systems Overview

With the development and subsequent refinement of XCS [4, 20, 21], Wilson introduced a new and unique classifier system utilizing a number of changes (many initially introduced with his ZCS [17]), which allowed XCS to achieve a high degree of success while also providing a more theoretically grounded and functionally transparent system. The fundamental details of XCS are described in this chapter. For a complete description of XCS parameters and implementation see [4, 6, 9, 20].

#### Parameter Updates

During execution XCS updates a variety of classifier parameters (*i.e.*,  $e_{cl}$ ,  $p_{cl}$ ,  $as_{cl}$ , and  $f_{cl}$ ) in order to estimate, or learn, their appropriate values using feedback from the system. Classifier parameters are updated in XCS using a two-phased process known as the MAM technique (*moyenne adaptive modifiée*) introduced by Venturini in [22].

Under the MAM technique a parameter is updated using a coarse-grained function at early stages and a more fine-grained one after a given amount of experience has been gained. In this way the parameter's value is quickly approximated early in its experience and then fine-tuned on subsequent iterations.

In XCS, when a classifier is relatively new (*i.e.*,  $exp_{cl} < 1/\beta$ ) the parameters (with the exception of fitness,  $f_{cl}$ ) are updated using the average of the experienced parameter values in order to quickly overcome any arbitrary initial value. The second phase of parameter updates is used when the classifier is sufficiently experienced (*i.e.*,

$exp_{cl} \geq 1/\beta$ ). In this second phase the updates use the Widrow–Hoff delta rule. This technique sums the current value of the parameter with the product of the learning rate,  $\beta$ , and the difference between the current value and the new value (e.g.,  $p_{cl} = p_{cl} + \beta(P - p_{cl})$ ).

It should be noted that the fitness parameter,  $f_{cl}$ , is always updated using the Widrow-Hoff technique, regardless of the classifier's experience level.

### Accuracy-Based Fitness

While similar to the traditional CS architecture in many ways, XCS differs in that the fitness of each classifier is determined by a function of its prediction accuracy relative to the accuracy of all other classifiers in the same GA niche. This accuracy-based fitness decouples the fitness measure from the credit-distribution method and focuses the GA on maximizing the accuracy of the population, regardless of the payoff magnitude attributed to each classifier.

In XCS the accuracy of a classifier is a function of the classifier's prediction error learned through experience with the environment. Since the initial introduction of XCS, the accuracy function has been refined to the following in order to improve sensitivity to small differences in error [6, 20]:

$$\text{IF } (\varepsilon_{cl} > \varepsilon_0) \tag{3.1}$$

$$k_{cl} = \alpha * (\varepsilon_{cl} / \varepsilon_0)^{-\nu}$$

ELSE

$$k_{cl} = 1$$

With this function, the accuracy of a classifier is dependent upon the ratio of its learned error in prediction,  $\varepsilon_{cl}$ , and the minimum error threshold parameter,  $\varepsilon_0$ . Should

the error of the classifier fall below the minimum error,  $\varepsilon_0$  (called the *accuracy criterion*), the classifier is considered to be completely accurate (*i.e.*,  $k_{cl} = 1$ ). If the error is greater than the minimum error, the classifier's accuracy is calculated as given with the fall-off rate ( $\alpha$ ) and power ( $\nu$ ) parameters dictating the degree of sensitivity in the calculation.

The fitness of a classifier is then determined by the ratio of the classifier's accuracy to that of all other classifiers in the same niche (*i.e.*, the classifier's *relative accuracy*) with the update following the Widrow-Hoff technique as described above.

That is, fitness is determined as follows:

$$f_{cl} = f_{cl} + \beta(k_{cl} / \sum k_{[A]} - f_{cl}) \quad (3.2)$$

### Niched GA

Another change implemented in XCS is the use of a niched GA. A *niched GA*, first introduced by Booker [8], operates on a subset of the overall classifier population rather than *panmictically* (*i.e.*, operating on the entire population) as in earlier CS implementations. Following work by Booker [4] the original XCS described by Wilson [4] placed the GA within the match set [M], that is, the set of all rules whose conditions were satisfied on the same clock cycle in which the GA was triggered. Subsequent work [20, 21] has indicated that placing the GA in the action set provides superior performance and this has since become standard in much of the XCS literature.

The use of a niched GA allows XCS to avoid the problem of greedy classifier selection by restricting parent selection to functionally related classifiers. This allows the parent selection process to operate without being impacted by any bias in the reward function as the candidates for GA selection are competing within the same reward niche.

### Credit Distribution

The third fundamental difference between XCS and a traditional CS lies in the credit distribution component. Whereas Holland's bucket-brigade algorithm is used to distribute credit from consumers to producers in traditional strength-based CS, the credit-distribution component in XCS uses a procedure patterned after the Q-Learning technique [18, 19] to update the payoff prediction of each classifier active on the previous clock cycle (*i.e.*,  $[A]_{i-1}$ ).

Q-Learning is a reinforcement learning (RL) technique [23] that has been found to be effective in learning a discounted estimate of the external reward anticipated on some subsequent time step from the classifiers activation on the current time step should the system follow the optimal policy (*i.e.*, choosing the actions with the maximum Q-values). The credit-distribution algorithm in XCS acts to propagate received rewards down through the activation chain, discounting them at each intervening time step. It was Wilson's goal to use Q-Learning to replace the traditional bucket-brigade with a more theoretically sound technique.

XCS utilizes a Q-Learning-like reinforcement algorithm when operating in multi-step environments. This RL technique updates the predictions of the classifiers in the action set activated on the previous clock cycle with the sum of the external reward received on the previous clock cycle and the maximum payoff predicted on the current clock cycle, discounted by the constant  $\gamma$  ( $0 \leq \gamma \leq 1$ ) as follows:

$$p_{[A]_{i-1}} = P_{i-1} + \gamma * \text{MAX}(p_{[A]}) \quad (3.3)$$

In this way the value of the discount factor determines the degree to which future payoff is considered in determining the appropriate action to take at the present time. It can be seen that though the discounting nature of the algorithm allows XCS to choose actions leading to some future payoff, the system may still erroneously choose a more appealing near term payoff should the discounting level be too low, or the number of intervening steps too long, for the long term payoff to appear advantageous.

Recently, it has been suggested that the role of the Q-Learning function in XCS has been misstated, as it accomplishes more than merely distributing credit amongst classifiers. Kovacs [24] suggests that the RL component (*i.e.*, the Q-Learning function) of XCS provides the learning aspect of the system while the GA, which has traditionally been described as the learning component, is responsible for generalizing between states. Furthermore, Kovacs [6] argues that XCS is a proper generalization of tabular Q-Learning, demonstrating the conceptual overlap between the two fields (*i.e.*, Q-learning and XCS) by discussing the minor modifications which, when made to XCS, produce a tabular Q-Learning system.

#### Exploration and Exploitation

During execution, XCS (and learning systems providing performance data in general) needs to achieve two goals: exploring the solution space and exploiting the learned solutions in order to test the system's performance. This tradeoff has come to be termed the explore/exploit dilemma. To facilitate a balance between these two goals, XCS is run in two distinct modes – exploration and exploitation [4, 25].

During exploration, the conflict-resolution component randomly chooses for activation an action set from the members of the match set generated by the current environmental input. This random selection ensures that all available action sets are evaluated with equal probability (assuming the encountered input states are relatively evenly distributed). As previously mentioned, it is only during exploration that the classifier's parameters are updated as this avoids biasing the system towards classifiers selected by the conflict-resolution function.

During exploitation the conflict-resolution process selects for activation the action set with the highest predicted payoff,  $p_{[A]}$ . The payoff predicted by each action set is determined by the fitness weighted predicted payoff of the set - where the fitness of each classifier in the action set is used to scale the degree to which its predicted payoff is used in determining the prediction for the set. This technique is shown below:

$$p_{[A]} = \frac{\sum (f_{cl} * p_{cl})}{\sum f_{cl}} \quad (3.4)$$

Wilson's original technique for allocating trials was to select exploration or exploitation based on an equal probability at the beginning of each iteration. As this has become standard in much of the XCS literature it is used exclusively in all experiments presented in this work.

### Covering

Introduced by Wilson [15] prior to the development of XCS, *covering* is used to add a new classifier to the system that matches the current state when the match set is found to be inadequate. Wilson's original XCS work [4] used covering in two situations: when no classifier existed matching the current state (*i.e.*, when the match set was empty)

or when the total prediction of the match set was less than the mean prediction of the population multiplied by a constant factor,  $\delta$ . A simplified covering trigger was later introduced [20], which created a new covering classifier when the number of actions in the match set fell below a threshold value,  $\theta_{mna}$ . The later form of covering trigger is used in all experiments presented in this work.

When creating a classifier through covering, the new classifier is generated with a random action not already present in the match set and its conditions are set to match (*i.e.*, cover) the current input state. Each allele in the classifier's conditions is then mutated at the probability  $P\#$  into a don't-care value (*i.e.*, a # bit). The new covering classifier is then added to the population, possibly triggering the deletion of an existing classifier if the population size is now greater than the upper limit,  $N$ .

The primary purpose of covering is to guarantee that the system takes some action in all encountered environmental states. By ensuring that each input is covered by some minimum number of classifiers, the system is forced to explore each encountered state and avoids having successful classifiers in another match set completely crowd out other under-explored niches.

A brief note should be made of the fact that Wilson advocates the use of a covering threshold set equal to the maximum number of actions possible. This results in the covering operator becoming the primary initial classifier discovery component and quickly develops a complete map while the GA focuses on the task of refinement (*i.e.*, generalization) of classifiers.

## Macroclassifiers

In an effort to make better use of system resources, Wilson [4] recognized that a population typically contains numerous duplicate classifiers. Through the use of a new classifier parameter, termed *numerosity*, XCS aggregates these individual duplicate classifiers, which Wilson called *microclassifiers*, into what Wilson called a *macroclassifier*.

With numerosity (*num*) a single instance of a unique classifier (*i.e.*, a macroclassifier) is used to represent every instance of that particular classifier (*i.e.*, the microclassifiers) by incrementing a macroclassifier's numerosity each time a copy of that classifier is added to the population, as a product of the GA or of the covering operator discussed below. Subsequently, when a classifier is selected for deletion from the system, its numerosity is decremented and the macroclassifier is removed from the population when its numerosity reaches zero as that indicates that no more instances of that classifier exist in the population. Work by Kovacs [9] has shown that the use of macroclassifiers has a minimal impact on system performance.

Within this document, the term 'population size' refers to the number of macroclassifiers within the population. The number of microclassifiers typically grows very quickly to the population size limit  $N$  and remains there for the duration of the run. One could argue that the population size limit should be tied to the number of macroclassifiers in the population as this better relates to the computational resources required for the system to operate.

### Subsumption Deletion

The process of *subsumption-deletion*, introduced by Wilson [21], acts to reduce the population size by identifying overly specific classifiers that are subsumed by other more general but equally accurate classifiers. In such a case the more specific classifier does not benefit the system as the more general classifier already accurately advocates its action, and does so in more match states. The process of subsumption-deletion takes two forms within XCS: GA-subsumption which occurs in the GA and action set subsumption which occurs in the action set.

In *GA-subsumption* the offspring of an accurate (*i.e.*,  $k_{cl} = 1.0$ ) and adequately evaluated classifier (*i.e.*,  $exp_{cl} > \theta_{sub}$ ) whose conditions match a set of states that are a proper subset of those matched by the parent's conditions and whose action is the same as that of the parent is considered to be *subsumed* by the parent. When the GA generates an offspring classifier that is subsumed by one of its parents it is discarded and the numerosity of the subsuming parent is increased by one.

The second form of subsumption, action set subsumption, takes place in every activated action set after the set is updated with the payoff received on that clock cycle. During action set subsumption the most general (*i.e.*, highest number of #'s) and accurate (*i.e.*,  $k_{cl} = 1.0$ ) classifier in the action set is identified and compared against each of the other members of the set. Each classifier that is subsumed is removed from the population and the subsuming classifier's numerosity is increased equivalent to the numerosity of the removed classifiers. As action set subsumption may result in the

removal of numerous established classifiers it is both more powerful and more dangerous than GA-subsumption.

Experimental results reported by Wilson [21] and Kovacs [9] indicate that while subsumption deletion does reduce population size it does not significantly affect system performance. However, informal comparisons performed by the author indicate that the value of subsumption may increase in size-constrained populations as its use allows a smaller population to test more candidate classifiers by opening up space in the population more quickly and thereby reducing the deletion pressure on fit population members.

#### Generality and Optimality of Classifiers

In advocating the use of accuracy-based fitness in conjunction with a niched GA, Wilson hypothesized that XCS should exhibit a tendency towards developing accurate and maximally general rules. *Maximally-general* rules are defined to be classifiers which are as general as possible while still remaining above a minimum accuracy threshold (*i.e.*,  $\epsilon_{cl} < \epsilon_0$ ). That is, they contain the maximum number of don't-care bits possible to accurately encode the set of match:action:payoff states masked by their conditions. This argument was encapsulated in what has been termed *Wilson's Generalization Hypothesis* [4], which states,

Consider two classifiers C1 and C2 having the same action, where C2's condition is a generalization of C1's. That is C2's condition can be generated from C1's by changing one or more of C1's specified (1 or 0) alleles to don't cares (#). Suppose that C1 and C2 are equally accurate in that their values of  $e$  are the same. Whenever C1 and C2 occur in the same action set, their fitness values will be updated by the same amounts. However, since C2 is a generalization of C1, it will tend to occur in more match sets than C1. Since the GA occurs in match sets, C2

would have more reproductive opportunities and thus its number of exemplars would tend to grow with respect to C1's (or, in macroclassifier term, the ratio of C2's numerosity to C1's would increase). Consequently, when C1 and C2 next meet in the same action set, a larger fraction of the constant fitness update amount would be "steered" toward exemplars of C2, resulting through the GA in yet more exemplars of C2 relative to C1. Eventually, it was hypothesized, C2 would displace C1 from the population.

Upon observing the tendency for XCS to develop maximally-general classifiers Kovacs later extended Wilson's Generalization Hypothesis with his *Optimality Hypothesis* [9], which states,

Given that, according to the Generalization Hypothesis, evolutionary pressure towards maximally general classifiers exists in XCS, an XCS system should eventually evolve a maximally general classifier for any payoff level which it is able to sample sufficiently.

As put forth by the generalization and optimality hypotheses the implicit nature of XCS' generalizing ability is important, as it is unique among classifier systems and allows XCS to develop compact populations of classifiers, with the degree of compactness dependent upon the degree of generality in the solution space. Results reported in [4, 6, 9, 21] as well as this work support both the generalization and optimality hypotheses.

Furthermore, in developing and evaluating the modifications proposed in this work, maintaining both the generalization and optimality hypotheses was a minimum requirement for all changes. To modify the system in such a way as to incapacitate its ability to develop maximally-general and optimal classifiers would be to obviate one of the most appealing aspects of the XCS framework.

## CHAPTER IV

### IMPROVING EXTENDED CLASSIFIER SYSTEM PERFORMANCE

This chapter presents modifications which, when made to an original XCS implementation, allow the system to achieve much higher performance in size-constrained systems. These modifications are intended to cause the system to make better use of the available population resources and to protect against the brittleness inherent in a solution represented by a small population of classifiers, where the removal of a single classifier can have a drastic impact on performance. Furthermore, these changes are presented in order to allow a more direct comparison between accuracy-based XCS and the effectiveness-based EXCS, presented in Chapter VI.

Experimental results presented at the end of this chapter compare the performance of XCS with a system utilizing the proposed modifications. These results agree with results reported for XCS [4, 6, 9, 26] as well as results reported for a variation of the proposed modifications [27] and demonstrate the performance improvement observed when using the proposed modifications in single-step and multi-step problems. Furthermore, the results show that a system utilizing the proposed modifications retains the desirable properties put forth in the Generalization Hypothesis and the Optimality Hypothesis.

## Proposed Modifications for Accuracy-Based Systems

### An Experience-Based Genetic Algorithm Trigger Function

The first proposed modification to the standard XCS implementation [4, 20] is made to the trigger function for the genetic algorithm. Wilson's original trigger algorithm attempts to evenly distribute GA activations across all encountered match/action sets regardless of the frequency with which those input states are encountered. Evenly distributing GA activations is achieved by triggering the GA when the average amount of time (*i.e.*, exploration steps) since the last GA activation in the given set of classifiers exceeds a given threshold value,  $\theta_{ga}$ . Note that as previously mentioned, the GA was later moved to operate on the action set as this was reported to provide better performance [21].

While Wilson's trigger algorithm does ensure that GA activations are distributed relatively evenly across the encountered match states (assuming the match states are 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. Therefore, it is quite common under the original algorithm for the GA to be triggered on action sets every time that set is encountered, even though many or all of the classifiers in the set are under-evaluated.

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 (*e.g.*, deletion of a valuable classifier) become more costly. Though the trigger threshold of the original algorithm,  $\theta_{ga}$  can be increased significantly to overcome this, it becomes difficult to determine the appropriate trigger level for a given population size due to its experience independent nature. Furthermore, this increased threshold 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 due to the increased threshold level.

The author proposes the use of a simple experience-based trigger algorithm to address the aforementioned shortcomings in size-constrained systems. Under this alternate algorithm, the GA is only triggered when the given set (match or action, depending upon placement of the GA) contains any number of classifiers with experience since the last GA trigger on that set greater than the trigger threshold. 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.

#### Focused Classifier Deletion

The second change proposed to the original XCS implementation is made to the deletion-selection algorithm. In XCS, a classifier is removed from the population whenever the population size (*i.e.*, the number of microclassifiers) exceeds the specified maximum size limit,  $N$ . 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, one independent of a classifier's fitness and one using fitness in the vote calculation. Kovacs [26] later combined the two into his 't3' algorithm that uses the experience level of the classifier to determine if fitness should be used in determining its deletion vote. We utilize this combined function in this work, as it appears to offer greater performance [26] and was utilized in the most recent XCS implementation described by Butz and Wilson [20].

The deletion-vote 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 indicates the number of identical classifiers in the population represented by the given classifier, and the learned action set size estimate represents an estimate of the number of classifiers present in the action sets to which the classifier belongs. Furthermore, if the classifier's experience is greater than the deletion threshold (*i.e.*,  $exp_{cl} > \theta_{del}$ ) and its fitness is less than a constant fraction ( $\delta$ ) 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. Therefore, the deletion algorithm calculates a classifier's deletion vote as follows:

$$vote_{cl} = num_{cl} * as_{cl} \quad (4.1)$$

$$IF ((exp_{cl} > \theta_{del}) \text{ AND } (f_{cl} / num_{cl} < \delta * (\sum f_{pop} / \sum num_{pop}))) \quad (4.2)$$

$$vote_{cl} = vote_{cl} * (\sum f_{pop} / \sum num_{pop}) / (f_{cl} / num_{cl})$$

Once the deletion vote of each classifier is determined, the deletion-selection algorithm selects a classifier for removal from the population. XCS' deletion-selection algorithm uses *roulette selection* where a classifier is probabilistically selected for

deletion based on the value of its deletion vote. While roulette deletion-selection is adequate in environments with a large population size, in more size-constrained environments its use can negatively affect system performance as its probabilistic nature often results in the deletion of effective and useful classifiers.

The author proposes 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.

Due to the use of max-deletion, the system maintains an implicit downward pressure on the numerosity of classifiers due to deletion-vote function factors of numerosity and action set size estimate (which is largely composed of a classifier's numerosity in size-constrained systems). In order to further increase the downward pressure on classifier numerosities the deletion vote function is modified by raising the numerosity factor by the new constraint measure,  $\Omega$ , which is an estimate of the degree of size constraint in the population for the given problem and population size. Therefore the first step of the new deletion vote function is given by:

$$vote_{cl} = num_{cl}^{\Omega} * as_{cl} \quad (4.3)$$

For all experiments reported in this work the value of the constraint measure,  $\Omega$ , was determined to be one if the size of the search space,  $S$ , was less than or equal to the size of the population; otherwise, the constraint measure is determined by a function

of the ratio of the search space to the population size. The constraint measure function utilized in this work is therefore:

$$\begin{aligned} &\text{IF } (S \leq N) && (4.4) \\ &\quad \Omega = 1 \\ &\text{ELSE} \\ &\quad \Omega = 1 + \ln((S / N)^{1/\beta}) \end{aligned}$$

Using this measure a population with a size which is greater than or equal to the size of the search space yields a constraint of  $\Omega=1$ , as it is operating without size-constraint. As the size of the population falls below the size of the search space the constraint measure grows. Though the constraint measure could (and probably should) include other factors taking into account such things as the amount of generalization possible in the search space (and therefore the size of the final solution set), this a-priori knowledge may not be readily apparent and is not assumed to be available in determining the appropriate value of  $\Omega$ . Note that the given function was chosen with little effort taken to evaluate alternative functions, though the constraint values returned by this function appear adequate for the configurations tested to date.

While maintaining downward pressure on classifier numerosities may slow the system's learning rate by diminishing the chances for fit classifiers to reproduce (which is addressed by the next modification) it is found to be advantageous in size-constrained systems 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 for such classifiers. Second, low classifier numerosities allow the smaller population of a size-constrained system to maintain a more diverse population of classifiers.

In practice, this change tends to result in all classifier's having a low numerosity until the population begins to condense on the optimal set of classifiers through subsumption. While this causes the initial population size of a modified system to be larger than an unmodified XCS (*i.e.*, more macroclassifiers), this allows a smaller population to evaluate a wider range of classifiers than is possible without the proposed modification while still developing a compact population once the maximally general classifiers have been discovered.

#### Focused Genetic Algorithm Parent Selection

The third proposed modification is to intensify the GA parent-selection probabilities. In a traditional XCS, the GA probabilistically selects the parents for the generation of new offspring from the current set (action or match, depending on placement of the GA) with the probability based on the value of each classifier's fitness.

For size-constrained systems, the selection probability was intensified by using the fitness of each classifier raised to the power of the constraint measure,  $\Omega$ , as given by:

$$\text{prob}(\text{cl}) = f_{\text{cl}}^{\Omega} / \sum f_{[\text{A}]}^{\Omega} \quad (4.5)$$

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.

Though reducing the probability that lower fitness classifiers will be selected for reproduction clearly narrows the scope of the search provided by the GA, this appears to be beneficial in small populations as the likelihood of wasting GA invocations and population space on inferior classifiers is reduced. In addition, this intensified selection probability helps to ameliorate the diminished favoritism of fit classifiers caused by the generally lower numerosities resulting from the previously described modification (*i.e.*, max-deletion with an intensified numerosity factor).

#### Parameter Updates on Exploit 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, to being updated during exploration steps only. This tends to evenly distribute parameter updates over the population due to the random nature of exploration steps.

The fourth modification proposed relates to the timing of parameter updates. This modification consists of simply updating a classifier's prediction and error estimates on both exploration and exploitation steps. The reasoning behind this change is two fold.

First, the information (*i.e.*, prediction estimate and error) gained from exploitation steps is typically lost in XCS. However, this information is valuable so long as it does not negatively bias the system towards classifiers chosen on exploitation steps, particularly so in a size-constrained system where the limited population size reduces the learning rate of the system. Second, by not updating the classifier experience count on

exploitation steps, the GA is not biased towards those classifiers typically chosen for their higher payoff predictions.

It should be noted that a slight bias is inherent in the fact that accurate classifiers chosen on exploitation steps will have more chances to refine their prediction error and will therefore tend to have higher accuracies than classifiers not generally chosen during exploitation. While this may violate the apparent XCS maxim of, “all classifiers are created equal” – it has been found to be advantageous in the problems evaluated to date, with both large and small populations, as it only results in a bias towards higher payoff garnering classifiers.

### Experimental Evaluation of Proposed Modifications

For the remainder of this chapter, experimental results that demonstrate the ability of XCS to learn an accurate and optimal match:action:payoff mapping for both the 6-Multiplexer and Woods-2 problems [4] are presented. In addition, results supporting the use of the proposed modifications in size-constrained environments are reported.

#### Methodology

All data gathered for this work used a version of Barry’s JXCS classifier system API [28] modified by the author to follow the most recent XCS implementation, as described in [20]. Further changes were made to incorporate the proposed modifications.

System parameters for all experiments reported in this work are as in [4] (see Appendix A) except where noted. The covering threshold parameter was set to require 1

classifier per match set (*i.e.*,  $\theta_{mna}=1$ ). This parameter value yields better performance in size-constrained systems and better tests the classifier discovery ability of the systems than does the level proposed by Wilson (*i.e.*, setting  $\theta_{mna}$  equal to the number of possible actions) [20]. The conservative parameter update order was followed [5] with the order being: experience, error, prediction, and then fitness.

Subsumption deletion was used in all experiments, though a new subsumption experience threshold was introduced,  $\theta_{sub[A]}$ , for use in triggering action set subsumption. As the negative impact of an erroneous action set subsumption can be far more severe than GA subsumption, a higher subsumption threshold was required. For all experiments reported in this work the value of the action set subsumption threshold was set to ten times the size of the input space (*i.e.*,  $\theta_{sub[A]}=10* \#inputs$ ).

This increased threshold level was found to reduce erroneous action set subsumptions caused by the occasional occurrence of an overgeneral classifier, which has yet to be deleted, encountering only a series of input states that it correctly maps to actions. This causes the overgeneral to momentarily appear to be accurate enough to subsume another, more truly accurate classifier. By raising the subsumption threshold the system is given more time to identify and eliminate overgeneral classifiers.

## 6-Multiplexer

The multiplexer class of problems are commonly used in the XCS literature as one of the benchmark experiments with which to test and compare classifier systems [4, 5, 6, 26]. Multiplexer problems operate on a bit string of length  $L=k+2^k$ , with  $k>0$ , and therefore provide a scalable problem easily encoded by a classifier system as bit-strings.

The 6-multiplexer (6-Mux) problem is a single step classification task in which the system must learn to map a 6-bit (*i.e.*,  $k=2$ ,  $L=6$ ) input string to one output bit. As with the logic circuit of the same name, the multiplexer operates by mapping two of the six input bits (*i.e.*, the address bits,  $k$ ) to one of the remaining four (*i.e.*,  $2^k$ ) input bits and placing the mapped bit value on the output bit (*i.e.*,  $F_6$ ). In disjunctive normal form the solution for the 6-multiplexer is given by (primes indicate negation):

$$F_6 = x_0'x_1'x_2 + x_0'x_1x_3 + x_0x_1'x_4 + x_0x_1x_5 \quad (4.6)$$

The multiplexer class of problems are frequently used as they present a challenging single step task with a scalable difficulty level 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 may 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 requires 128 unique classifiers to be complete (*i.e.*,  $2^6 = 64$  unique inputs with two possible actions each) a smaller set of 16 optimal classifiers can accurately cover the entire input space given a two-level payoff landscape.

For all multiplexer results reported in this work, the system inputs were randomly generated 6-bit strings. Rewards of 1000 and 0 were returned to the system for right and wrong answers respectively.

Performance Measures. The results presented in this work for the 6-Mux problem use two different performance measures commonly reported in the literature.

The first measure, simply called *performance*, is a moving average of the percentage of the last 50 exploit steps sampled that earned the maximum payoff possible (*i.e.*, that gave the right answer for the given 6-bit input string). *Optimality*, the second measure of system performance, shows the percentage of the optimal set of classifiers present in the population on any given sampled iteration. As stated, for the 6-Mux problem with two payoff levels, the optimal subset consists of 16 classifiers.

$N=400$ . Figures 2 and 3 compare the performance and optimality respectively of two XCS systems in the 6-Mux environment with a population size of  $N = 400$ . The

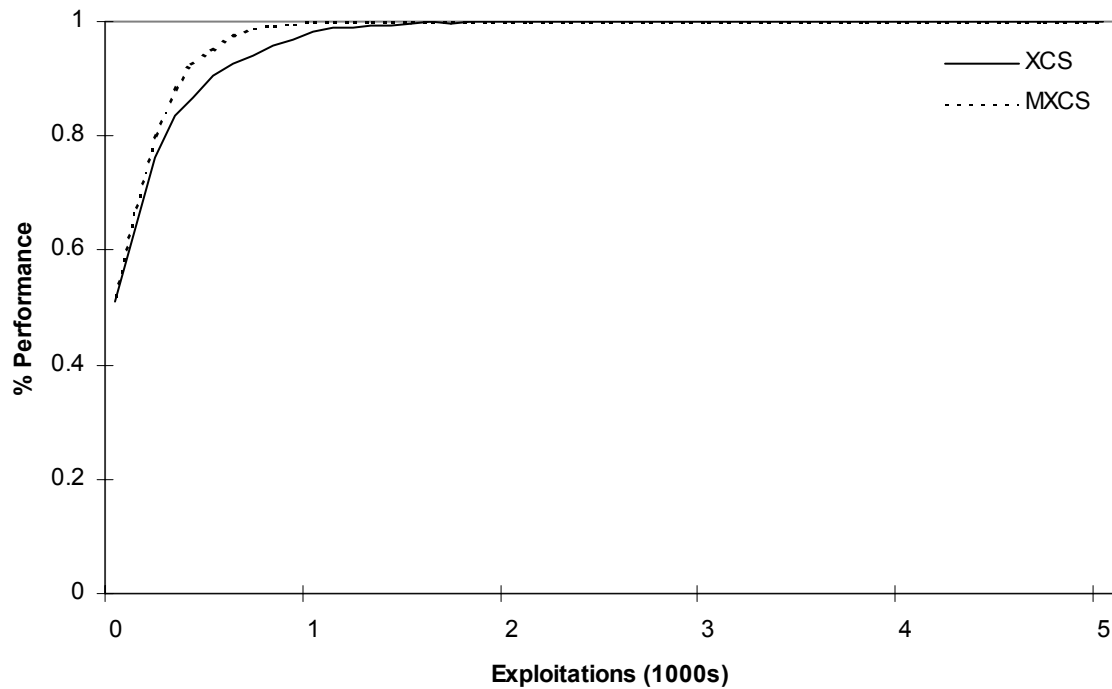


FIGURE 2. Performance of XCS and MXCS in 6-Mux with  $N=400$  and System Configurations as in Table A-1. Curves are Averages of 100 Runs Sampled Every 50 Exploitations

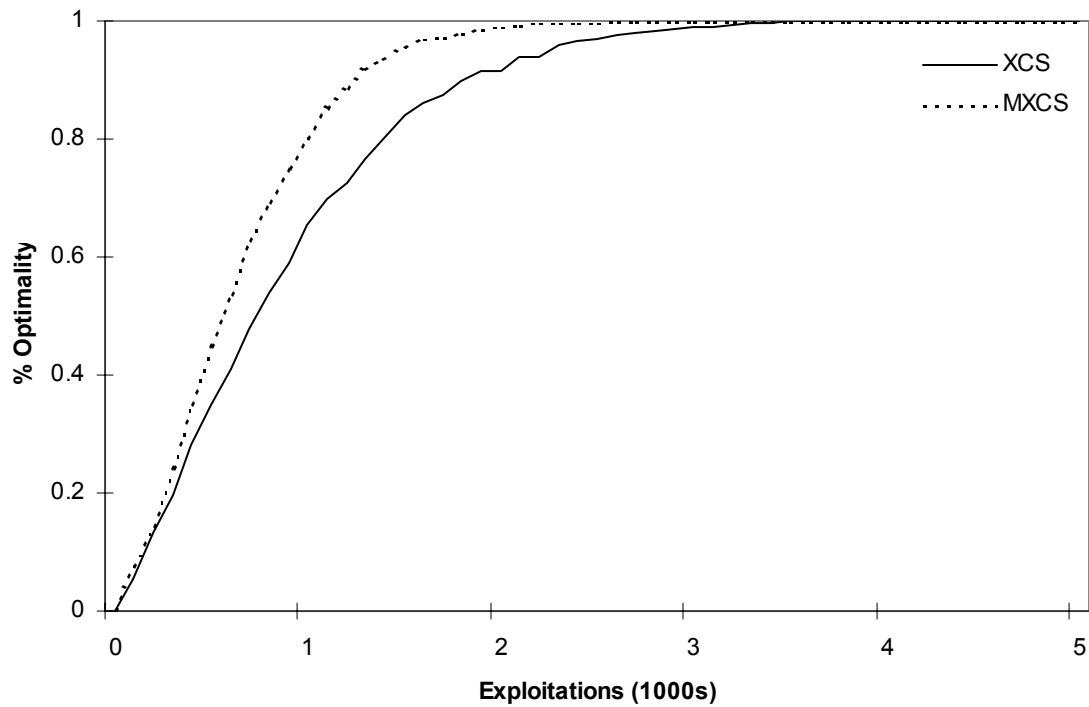


FIGURE 3. Optimality of XCS and MXCS in 6-Mux with  $N=400$  and System Configurations are as in Table A-1. Curves are Averages of 100 Runs Sampled Every 50 Exploitations.

first system implements the original XCS as described in [4, 20], while the second system implements a modified XCS (MXCS) utilizing the modifications proposed in this chapter.

Note that the GA trigger threshold is set to zero (*i.e.*,  $\theta_{ga}=0$ ) for both systems in order to allow a fair comparison, as the different GA triggering functions implemented in the two systems cause XCS to invoke the GA far more often than MXCS for a given threshold value. The chosen threshold level of  $\theta_{ga}=0$  zero resulted in no apparent degradation of performance. Hence, setting  $\theta_{ga}=0$ , and the results reported, highlights the

apparent irrelevance of the standard threshold value (*i.e.*,  $\theta_{ga}=25$ ) used in the literature for 6-Mux problems with  $N=400$ .

As given by equation 4.4, the size-constraint measure,  $\Omega$ , for MXCS with  $N=400$  was determined to be  $\Omega = 1$ , as the 6-Mux problem with  $N=400$  has no size-constraint (*i.e.*, the search space size of  $S=128$  is smaller than  $N=400$ ). Therefore, the results reported for the  $N=400$  configuration utilize only two of the four proposed modification, max-deletion and updating classifier prediction and error estimates on exploitation and exploration episodes.

As shown in Figures 2 and 3, the modified system (MXCS) achieves slightly better initial performance and optimality than XCS - though the difference is minor. XCS achieves 100% performance on all runs after 2,500 exploitations while MXCS achieves comparable performance after 1,500 exploitations. Optimality of 100% was achieved by XCS and MXCS after 4,100 and 2,900 exploitations respectively.

These results indicate that, given a size constraint of  $\Omega=1$ , the applied modifications (*i.e.*, max-deletion and exploit episode parameter updates) have a moderately beneficial impact on performance. Furthermore, the results shown for XCS approximate those reported by Wilson [4] and Kovacs [5, 6, 26], which supports the assertion that the original experience-independent trigger function utilized by XCS has little impact on system performance at the typical threshold value.

Given the reported results for XCS with a GA trigger threshold of  $\theta_{ga}=0$ , it appears that the performance of XCS in 6-Mux, as reported in the literature, relies largely on the fact that the population size typically used (*i.e.*,  $N=400$ ) is large enough to make

up for any detrimental effects caused by the use of the original GA trigger algorithm (*e.g.*, generating offspring faster than they can be evaluated) and roulette deletion (*e.g.*, occasionally choosing fit classifiers for deletion). However, as demonstrated by the following experiment, these issues become problematic for XCS in size-constrained 6-Mux problems.

$N=40$ . Figures 4 and 5 show a comparison of the performance and optimality respectively of the same two systems with the population size reduced by an order of

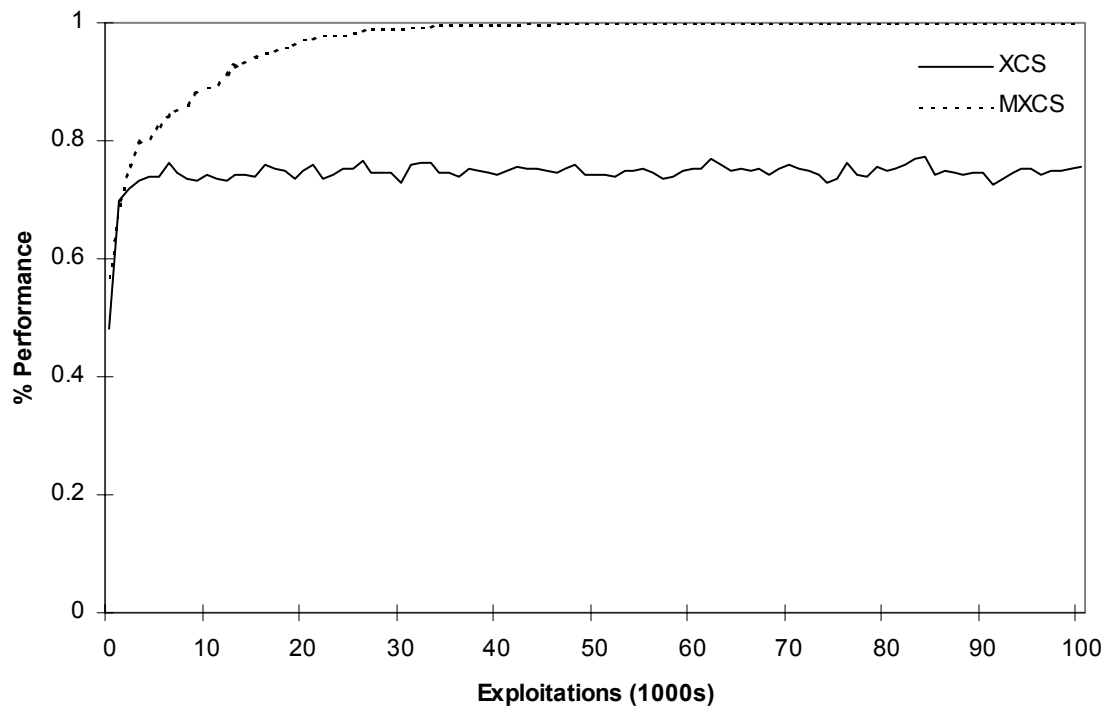


FIGURE 4. Performance of XCS and MXCS in 6-Mux with  $N=40$  and System Configurations as in Table A-1. Curves are Averages of 100 Runs Sampled Every 1000 Exploitations.

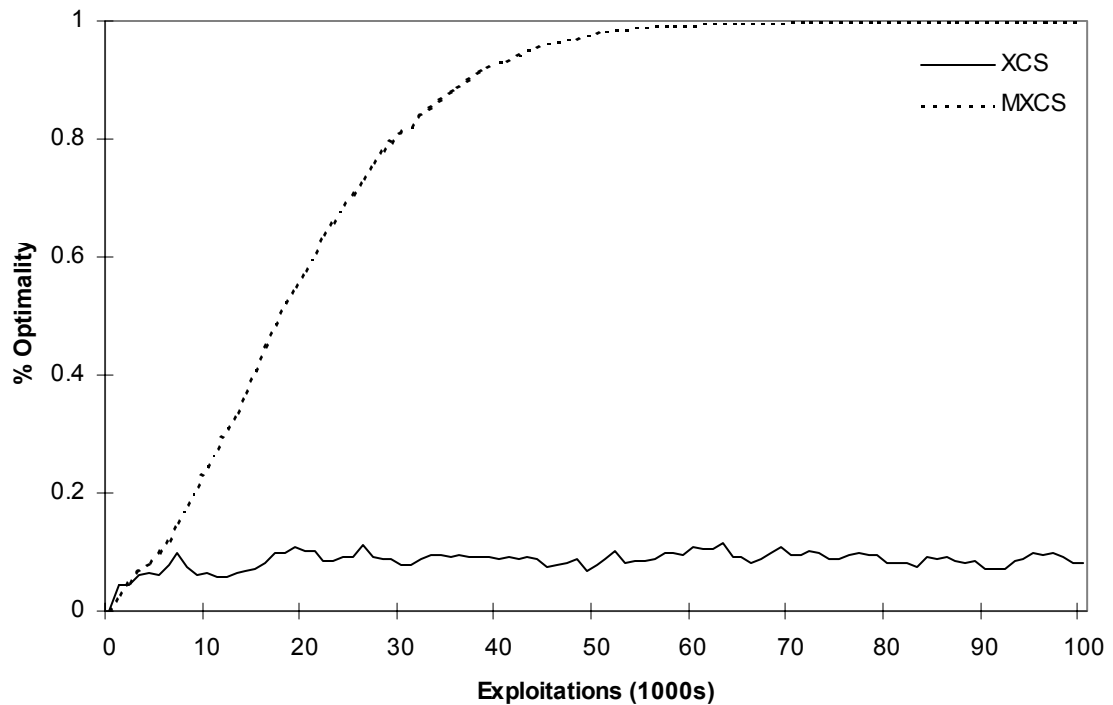


FIGURE 5. Optimality of XCS and MXCS in 6-Mux with  $N=40$  and System Configurations as in Table A-1. Curves are Averages of 100 Runs Sampled Every 1000 Exploitations.

magnitude to  $N=40$ . The GA triggering threshold has been changed for both systems in an effort to cause each to perform approximately the same number of GA invocations.

For XCS the GA trigger threshold was increased to  $\theta_{ga}=185$ , while MXCS' threshold was set to  $\theta_{ga}=25$ . These threshold values resulted in average GA invocations per run of 4,423 for XCS and 4,368 for MXCS.

As shown in Figures 4 and 5, XCS is incapable of achieving an average performance level higher than 75% or an optimality level higher than 10% - while the use of the proposed modifications allows MXCS to achieve 100% performance and optimality. Though numerous configurations were tested XCS appears to be incapable of

developing a successful population in the size-constrained system, regardless of the GA trigger threshold or the number of iterations evaluated.

Note that the number of iterations required for MXCS to reach these performance levels was increased significantly (by roughly two orders of magnitude) due to the reduced number of classifiers being evaluated at one time in the size-constrained configuration. These results also demonstrate that the system implementing the proposed modifications appears to have retained the desirable properties put for by the generality and optimality hypotheses, even when operating in a size-constrained configuration.

### Woods-2

Methodology. This section compares the performance of XCS and the modified XCS in the Woods-2 problem [4] at the traditional population size of  $N=800$  and a size-constrained configuration with  $N=80$ .

Woods-2, developed by Wilson [4], is a multi-step Markov problem with delayed rewards in which the system represents an “animat” [15], or organism, searching for food on a 30 by 15 grid. As shown in Figure 6, the Woods-2 environment 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” that 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 which are separated by empty space and the grid “wraps around” (*i.e.*, moving off the edge of the grid places the animat on the opposite edge of the grid), thereby creating a continuous environment.

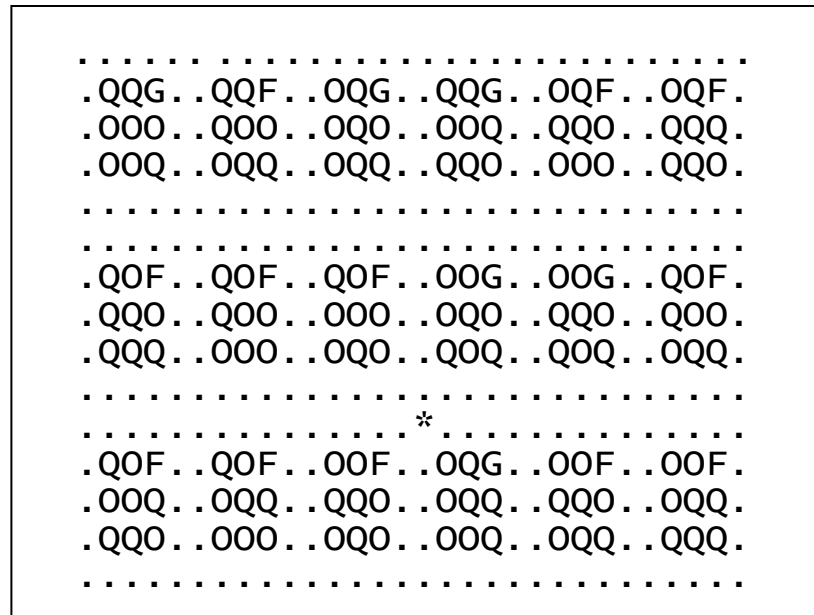


FIGURE 6. The Woods-2 Environment with Food Represented by "F" and "G", Rock Represented by "O" and "Q", Empty Space Represented by "." And the Animat Represented by "\*".

The system input is a 24-bit binary string representing the eight spaces surrounding the location currently occupied by the animat (*i.e.*, the animat's "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.

Due to the three-bit encoding scheme used for the five possible location types Woods-2 is capable of 70 unique input conditions [4]. With eight possible actions the search space size for the Woods-2 problem is therefore,  $8 \times 70 = 560$ .

Performance Measures. For Woods-2, performance is measured as a moving average of the number of steps required for the animat to reach food on the last 50 exploitation episodes. A performance level of 1.7 steps on average is the optimal solution for Woods-2 as no population of rules can do better [4].

$N=800$ . Figure 7 shows the performance of XCS and MXCS in the Woods-2 problem with a population size of  $N=800$ . As with the 6-Mux experiments previously described, the GA trigger threshold parameters were chosen in order to allow a fair comparison of XCS and MXCS along the exploitation axis. For Woods-2 with  $N=800$  the threshold for XCS was set to  $\theta_{ga}=25$  (in keeping with Wilson's experiments) while  $\theta_{ga}$  for

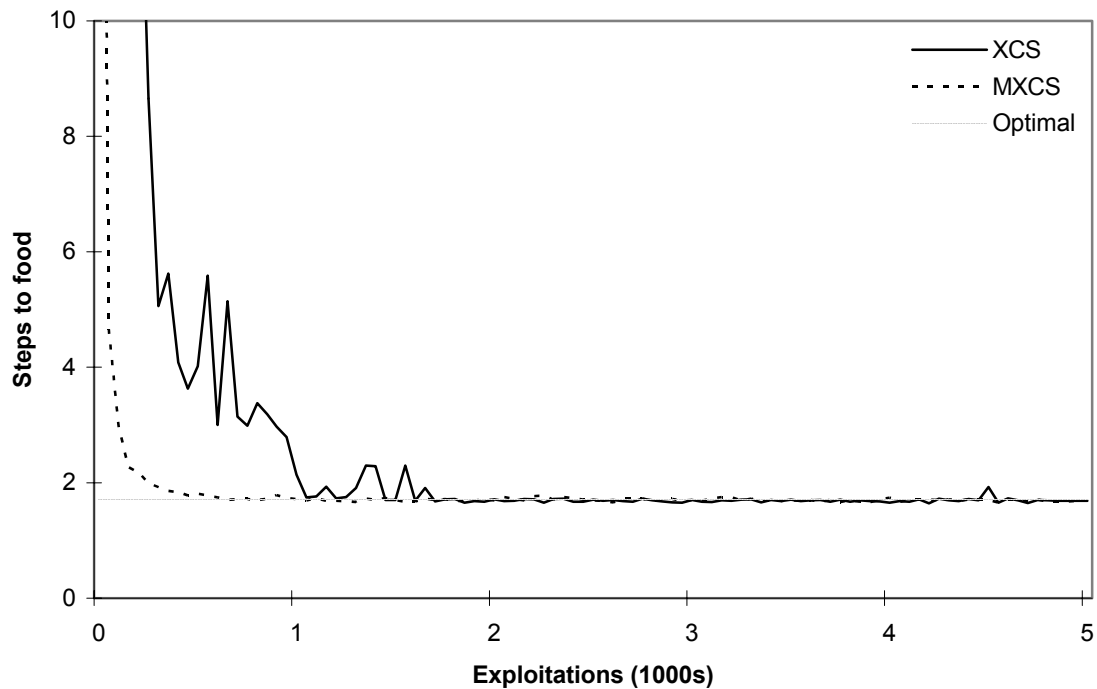


FIGURE 7. Performance of XCS and MXCS in Woods-2 with  $N=800$  and system Configurations as in Table A-2. Curves are Averages of 10 Runs Sampled Every 50 Exploitations.

MXCS was set to 3 (which was the lowest setting possible without degrading performance). These GA thresholds resulted in average GA invocations per run of 70.1k for XCS and 50.4k for MXCS, an apparent advantage for XCS.

While these results differ slightly from those reported by Wilson [4] and Kovacs (they reported optimal performance around exploitation episode 1,000 compared to approximately exploitation episode 1,500 shown in Figure 7) the difference in performance appears to be caused by the use of the action set subsumption (even with the increased threshold level), which was not used by Wilson or Kovacs, and the simplified covering operator (as proposed by Butz and Wilson [20]) with a single covering action threshold (*i.e.*,  $\theta_{mna}=1$ ). Wilson’s original Woods-2 experiments utilized a more “strength” biased covering operator where a covering classifier was generated when the mean payoff prediction of the match set,  $[M]$ , was less than a given fraction ( $\varphi$ ) of the mean population prediction (Wilson used a level of  $\varphi=0.1$ ).

As demonstrated by Figure 7, MXCS is able to quickly develop a successful population of classifiers for Woods-2 with  $N=800$ , showing that the proposed modifications, with  $\Omega=1$ , have not degraded the ability of the system to learn in a multi-step environment. It should be noted that the average population size (not shown) for MXCS tends to remain near the population maximum,  $N$ , until the population begins to condense through subsumption.

$N=80$ . Figure 8 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 [4]). The GA thresholds were selected to

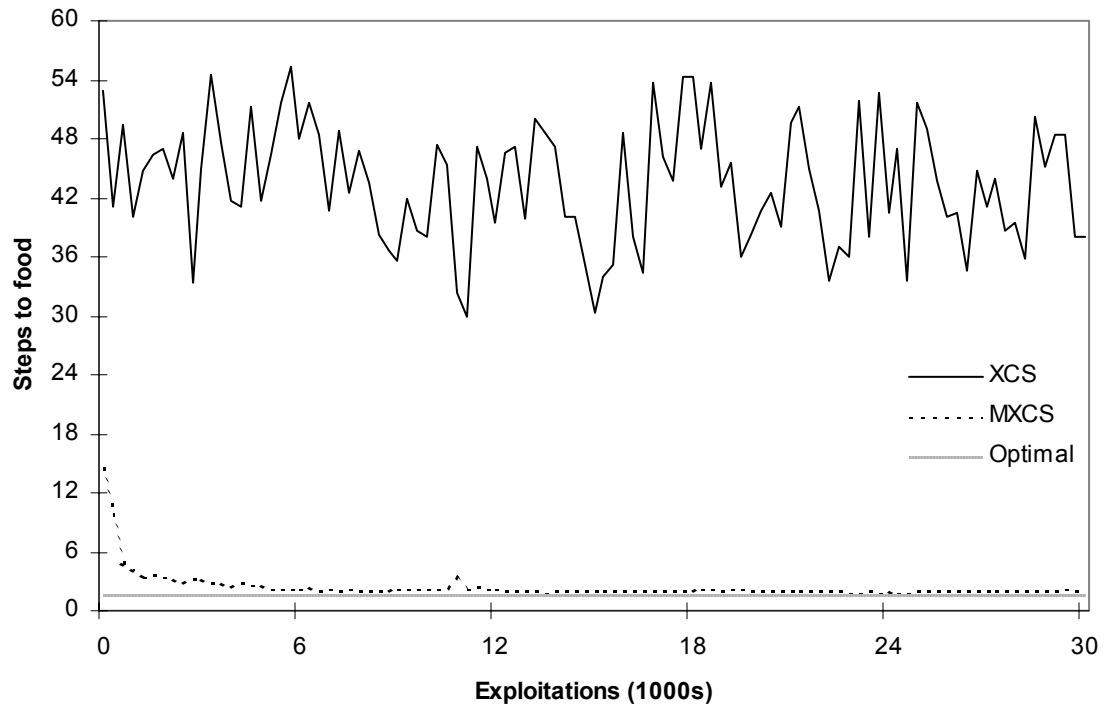


FIGURE 8. Performance of XCS and MXCS in Woods-2 with  $N=80$  and System Parameters as in Table A-2. Curves are Averages of 10 Runs Sampled Every 300 Exploitations.

produce approximately the same number of GA invocations by each system ( $\theta_{ga}=500$  for XCS and  $\theta_{ga}=25$  for MXCS).

As shown, the modified system is able to achieve a high level of performance while XCS is unable to develop a stable population of classifiers. Note though that by the end of the 30k exploitation episodes shown, the performance level for MXCS only reaches approximately 2.10 steps on average to reach food. While this is not the optimal solution (*i.e.*, 1.7), it is considerably better than the performance level of roughly 30 to 50 steps on average achieved by XCS. Furthermore, it is possible that MXCS may achieve higher performance under different configurations (*e.g.*, a different  $\beta$  value, a higher  $\theta_{ga}$

or more iterations) as the parameter settings used here were chosen to follow those used in [4], which may not be ideally suited for size-constrained system (*e.g.*, informal experimentation indicates that higher  $\theta_{ga}$  values offer better performance).

### Summary of Results

The results reported in this chapter have demonstrated the improvement in performance achieved in the 6-Mux and Woods-2 environments through the use of the four proposed modifications. These modifications consisted of: intensified GA selection and deletion-vote functions, using max-deletion selection and exploit episode updates of activated classifier's payoff prediction and error estimates.

With these modifications, the system is intended to make more cautious use of the available resources, resulting in improved performance, particularly in size-constrained configurations. Furthermore, these results show that the modified system appears to have maintained the inherent abilities of XCS to achieve accurate, complete and potentially optimal populations of classifiers as put forth by the generalization and optimality hypotheses.

CHAPTER V  
AN EFFECTIVENESS-BASED  
EXTENDED CLASSIFIER  
SYSTEM

Accompanying the introduction of his accuracy-based XCS system, Wilson presented an overview of the problems associated with strength-based fitness [4], with the discussion furthered by Kovacs in [5, 6]. While Wilson states that the majority of these issues can be overcome through the use of a niched GA, he states that the use of strength-based fitness still suffers from two problems.

First, the GA in a strength-based system cannot distinguish between two classifiers with the same strength, although the first may be accurate at receiving the payoff associated with it while the second may be sporadically receiving a higher reward but equivalent on average to the first. This problem is more broadly described as the problem of greedy classifier selection [5] discussed in Chapter I.

Second, while the use of don't-care symbols in their conditions allows classifiers to generalize over states, there appears to be in strength-based systems, as Wilson puts it, "... no clear tendency or, indeed, theoretical reason, for accurate generalizations to evolve." [4]

While XCS has been shown to be capable of learning complete and maximally general mappings of the payoff landscape [4, 5, 6, 24], this completeness becomes

problematic in systems with limited resources relative to the size of the search space. Clearly, as the size of the solution search space increases XCS requires a comparable increase in system resources in order to learn and maintain a complete map, regardless of the usefulness of the additional classifiers.

Though the ability of XCS to generalize over states allows it to eventually develop a compact map, potentially much smaller than the size of the search space (depending upon the degree of generalization afforded by the problem), the initial classifier discovery and generalization phases can require a significant percentage of the population size. As previously stated, Wilson recognized this problem and called for the development of techniques aimed at alleviating it [4].

EXCS addresses these issues by focusing system resources on accurate and rewarding classifiers. By combining accuracy, strength and specificity into the fitness measure, EXCS can operate with high performance in systems with constrained resources where XCS would be unsuccessful due to its need to develop a complete map. Though the map developed by EXCS is an incomplete one, the use of a niched GA and the accuracy factor in the effectiveness measure allow the system to avoid the problems previously described for purely strength-based systems.

This chapter defines a new effectiveness measure and the modifications required for its use in the XCS architecture. In addition, other modifications are introduced which allow the system to maximize the potential benefit in using an incomplete map. Experimental results are then presented showing the potential of

effectiveness as a fitness measure in environments with significantly limited resources and which do not require a complete map for an adequate solution.

### Modifications Required for Effectiveness- Based Extended Classifier Systems

While employing the same architecture and modifications as the MXCS presented in Chapter V, EXCS does require further modifications to the accuracy, fitness and conflict-resolution functions in order to operate effectively. The following sections describe these changes. In addition, a new culling operator is introduced in order to speed the discovery of highly effective classifiers. It should be noted that while numerous configurations were tested, these modifications did not appear to be beneficial in accuracy-based systems such as XCS or MXCS.

#### Averaging Accuracy Function

In developing EXCS it was found that the standard accuracy function used in XCS was too sensitive to near term changes in prediction error and too insensitive to the classifier's long term accuracy history to be used in determining the effectiveness of classifiers. This is particularly problematic in systems employing max-deletion due to the less forgiving nature of the algorithm. In an effort to improve the accuracy function for use in EXCS, the author employed an accuracy averaging function with the goal of better encoding the accuracy history of a classifier.

The *accuracy averaging function* presented here achieves this goal in EXCS by using two factors to calculate a classifier's accuracy. First a time sensitive average of the classifier's accuracy is determined. This is calculated by taking the ratio of the sum of

the classifier's accuracy over time,  $k\_sum_{cl}$ , and its experience as measured by,  $k\_exp_{cl}$ , both discounted by a constant fraction,  $\psi$ , as shown in equations 5.1 and 5.2. Second, this new discounted accuracy average and the standard classifier accuracy,  $k_{cl}$ , are averaged as shown in equation 5.3.

$$k\_sum_{cl} = (k\_sum_{cl} * \psi) + k_{cl} \quad (5.1)$$

$$k\_exp_{cl} = (k\_exp_{cl} * \psi) + 1 \quad (5.2)$$

$$k\_ave_{cl} = ((k\_sum_{cl} / k\_exp_{cl}) + k_{cl}) / 2 \quad (5.3)$$

With the given algorithm the weight attributed to the accuracy recorded  $n$  iterations ago used in determining the average accuracy is given by  $\psi^n$ . For all experiments reported in this work the value of the accuracy discount rate was set to  $\psi=.999$ . Though a variety of functions were evaluated for determining the value of  $\psi$ , the given value was chosen due to its simplicity and apparent effectiveness in all problems evaluated to date.

This new accuracy measure,  $k\_ave_{cl}$ , has been found to be effective in encoding both the near term and long term accuracy of classifiers. The immediate accuracy factor (*i.e.*,  $k_{cl}$ ) causes the average accuracy to react quickly to recent changes in the classifier's prediction error while the factor of the classifier's long-term accuracy history (*i.e.*,  $k\_sum_{cl}/k\_exp_{cl}$ ) helps to identify and differentiate between classifiers with occasionally inaccurate predictions. The use of the discount rate,  $\psi$ , allows the accuracy history encoded in the averaging accuracy measure to adapt over time to changes in the environment while weighing the most recently recorded accuracies more heavily than significantly earlier ones.

### Effectiveness-based Fitness Calculation

Similar to the effectiveness measure presented in Booker [8], which was a function of a classifier’s “impact” (*i.e.*, payoff), “consistency” (*i.e.*, prediction error) and “match score” (*i.e.*, specificity), this work measures classifier effectiveness as the product of a classifier’s predicted payoff ( $p_{cl}$ ), accuracy average ( $k_{ave_{cl}}$ ) and generality ( $gen_{cl}$ ) – where *generality* is the percentage of the classifier’s condition consisting of don’t-care bits (#’s). Therefore, the effectiveness of an individual classifier is given by:

$$eff_{cl} = p_{cl} * k_{ave_{cl}} * gen_{cl} \quad (5.4)$$

The fitness calculation in EXCS then proceeds as in XCS with two differences. First, the GA in EXCS operates on the match set niche, [M], as in the original XCS proposed by Wilson [4]. This is done in order to use the GA to drive the system towards a small number of effective actions per match set. Second, where XCS calculates a classifier’s fitness to be the portion of the niche’s accuracy sum attributed to that classifier, EXCS calculates fitness as the ratio of a classifier’s effectiveness to the maximum effectiveness found in the niche (*i.e.*, the match set), as shown below:

$$f_{cl} = eff_{cl} / \text{MAX}(eff_{[M]}) \quad (5.5)$$

The new fitness measure of equation 5.5 prevents the fitness of each classifier from being effected by the number of other classifiers in the same match set, which can be troublesome when comparing classifier fitness across GA niches, as in the deletion-vote calculation. The use of deletion-selection to evenly distribute classifiers among the action sets was a goal of Wilson’s in developing XCS. However, this has been found to be detrimental to performance in systems employing max deletion with the GA in [M] as

highly effective classifiers in crowded GA niches (as is seemingly common when the GA is in [M]) have higher deletion votes than less effective classifiers in more sparsely populated niches.

Though moving the GA and fitness calculation to the action set would help to prevent greedy classifier selection, this would result in a complete map (or at least a complete reward-garnering map if the culling operator described later in this chapter is used to remove zero payoff classifiers). While developing a complete map is ideal when possible, these modifications are proposed under the assumption that the system is operating in a problem / system configuration, which makes a complete map difficult or impossible to achieve.

It should also be noted that the action set size estimate,  $as$ , was changed for EXCS to measure the average size of the match sets in which the classifiers are found, rather than the action sets as in XCS. This peer group measure helps the system to avoid the disruptive impact of greedy classifier selection by the GA.

### Conflict-Resolution

In order to accommodate the changes introduced in EXCS, the conflict-resolution component must be modified to make up for the lack of a complete map in selecting actions for activation. XCS takes great advantage of the fact that accurate predictions of low payoff actions can be used to direct action selection towards higher payoff actions during conflict-resolution on exploitation steps. These low-payoff but accurate classifiers exist in the population thanks to the complete payoff map. EXCS, on the other hand, is at a disadvantage as the incomplete map developed results in

inexperienced or inaccurate classifiers with high predictions appearing favorable to the conflict-resolution function due to the lack of accurate classifiers in that action set (*e.g.*, the fitness weighted prediction of a classifier in a single member action set is independent of the classifier's fitness, and therefore accuracy).

This problem is overcome by introducing a new factor into the conflict-resolution function that uses a classifier's experience in calculating the estimated payoff prediction for an action set. This is implemented by simply using the square of a small fraction (*i.e.*, 1%) of the predicted payoff and the same small fraction of the fitness of the classifier in calculating the fitness weighted prediction average for the action set - until the classifier's experience becomes greater than the constraint measure times the deletion threshold (*i.e.*,  $exp_{cl} < \Omega * \theta_{del}$ ), at which time the full prediction is used. Therefore, the new algorithm for calculating the contribution of an individual classifier to the fitness-weighted payoff prediction for an action set is as follows:

$$\text{IF } (exp_{cl} < \Omega * \theta_{del}) \quad (5.6)$$

$$\Sigma (p * f)_{[A]} = \Sigma (p * f)_{[A]} + .01^2 * p_{cl} * f_{cl}$$

$$\Sigma f_{[A]} = \Sigma f_{[A]} + .01 * f_{cl}$$

ELSE

$$\Sigma (p * f)_{[A]} = \Sigma (p * f)_{[A]} + p_{cl} * f_{cl}$$

$$\Sigma f_{[A]} = \Sigma f_{[A]} + f_{cl}$$

In this way, inexperienced classifiers with high but potentially inaccurate predicted payoffs are unable to deceive the action selection process. Using the fraction of the classifier's fitness (*i.e.*,  $0.1 * f_{cl}$ ) in the fitness sum for the action set prevents the

inexperienced classifier from distorting the payoff predicted for that action set. The use of the square of the fraction in the payoff prediction (*i.e.*,  $.01^2 * p_{cl} * f_{cl}$ ) helps to prevent the inexperienced classifier from overestimating its predicted payoff relative to the rest of the action set as well as protecting against situations where the classifier is the only member of the action set.

### Population Culling

The final modification made in EXCS is the introduction of a new deletion operation, termed *culling*. The purpose of the culling operation is to rapidly remove ineffective classifiers from the population. This is accomplished by immediately removing classifiers from the population when the classifier's experience is greater than the deletion threshold ( $exp_{cl} > \theta_{del}$ ), its effectiveness is below the minimum error parameter,  $\epsilon_0$ , and the percentage of the population composed of ineffective classifiers is above a given threshold,  $\theta_{cull}$ .

The threshold of  $\theta_{cull}$  is used as it was found to be beneficial to performance to leave some number of "easy targets" in the population for the deletion selection process. Otherwise, should all ineffective classifiers be removed, the deletion-selection process tends to be forced to remove effective classifiers, as the bulk of the population consists of experienced effective classifiers and inexperienced classifiers with low deletion votes due to their low experience level.

While these ineffective classifiers would be removed through normal operation of the GA, the slow rate of removal results in ineffective classifiers taking up a significant amount of the system's resources, both in evaluations and population space.

With the removal of these classifiers it is assumed that the random search provided by the covering operator will more quickly find an acceptable classifier than would be achieved through operation of the GA.

It should be noted that an additional factor would be required in the culling function for systems where classifier conditions may not match any encountered environmental states. These “matchless” classifiers would never be present in any selected action sets and thus would never receive any experience. Therefore, under the proposed deletion algorithm (*i.e.*, max-deletion) these unused classifiers would linger in the system due to their experience always being below the deletion threshold ( $\theta_{del}$ ) and their deletion vote always being a function of their low numerosity and peer set size estimates. As EXCS is focused on removing ineffective classifiers it might be advantageous to add a classifier parameter measuring the time since the classifier was last matched, and culling classifiers exceeding some “idleness” threshold.

### Experimental Comparison of Accuracy- Based and Effectiveness-Based Systems

This section presents experimental results showing that the effectiveness-based fitness measure implemented in EXCS offers superior performance compared to MXCS, when system resources are tightly constrained in the examined problems.

#### Methodology

The first experiment presented compares the performance of MXCS and EXCS in the 6-Mux environment with constrained population sizes of  $N=40$ , and  $N=20$ .

Results for  $N=400$ , though not shown, indicate that EXCS and MXCS offer essentially equivalent performance in large populations.

In order to demonstrate the ability of EXCS to learn effective mappings of a multi-step problem domain, results are then presented comparing the performance of MXCS and EXCS in the size-constrained Woods-2 environment with population sizes of  $N=80$  and  $N=40$ . Again, the performance of the two systems was found to be equivalent in the unconstrained Woods-2 problem with  $N=800$  and is therefore omitted.

### 6-Multiplexer

Performance Measures. The performance measures used in the 6-Mux results presented in this chapter are the same as in Chapter V, with the exception of the set of classifiers used in calculating the optimality measure. As previously discussed, a traditional XCS implementation would consider the 6-Mux problem with two payoff levels to contain 16 optimal classifiers – 8 for each of the two payoff levels. Due to the nature of EXCS to discard the “ineffective” half of the optimal set, the set of optimal classifiers measured in this chapter consists of only those classifiers which are optimally general and garner a reward (*i.e.*,  $p_{cl} = 1000$ ), that is, they are *optimally effective*. The set of optimally effective classifiers of a two payoff level 6-Mux problem contains 8 unique classifiers. Therefore, for the sake of comparison the optimality measure used here disregards the 8 classifiers that earn no payoff but are indeed optimally general.

$N=40$ . Figures 9 and 10 show the performance and optimality respectively of EXCS and MXCS in a size constrained system with a population size of  $N=40$ , as was used in Chapter V. Note that for EXCS the deletion-vote mean fitness threshold fraction,

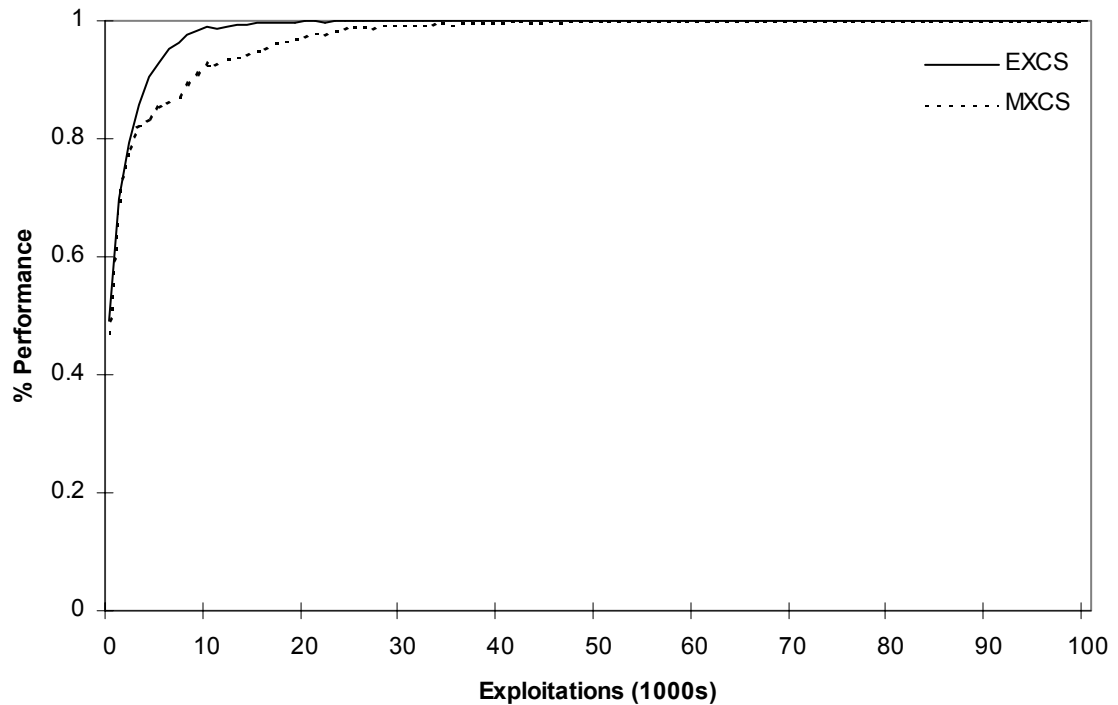


FIGURE 9. Performance of EXCS and MXCS in 6-mux with  $N=40$  and system Parameters as in Table A-3. Curves are Averages of 100 Runs Sampled Every 1000 Exploitations.

$\delta$ , has been ignored (*i.e.*, raised to  $\delta=\infty$ ) in order to cause the deletion-vote function to factor the fitness of all sufficiently experienced classifiers into their deletion vote. This was found to offer superior performance in EXCS compared to the standard value,  $\delta=0.1$ , possibly owing to the fact that the effectiveness-based fitness measure in EXCS contains more information (*i.e.*, generality, prediction magnitude and accuracy) and therefore offers a wider range of useful values.

As shown in Figure 9 the effectiveness-based system (EXCS) is able to achieve 100% performance on all runs roughly 23k exploitations earlier than MXCS

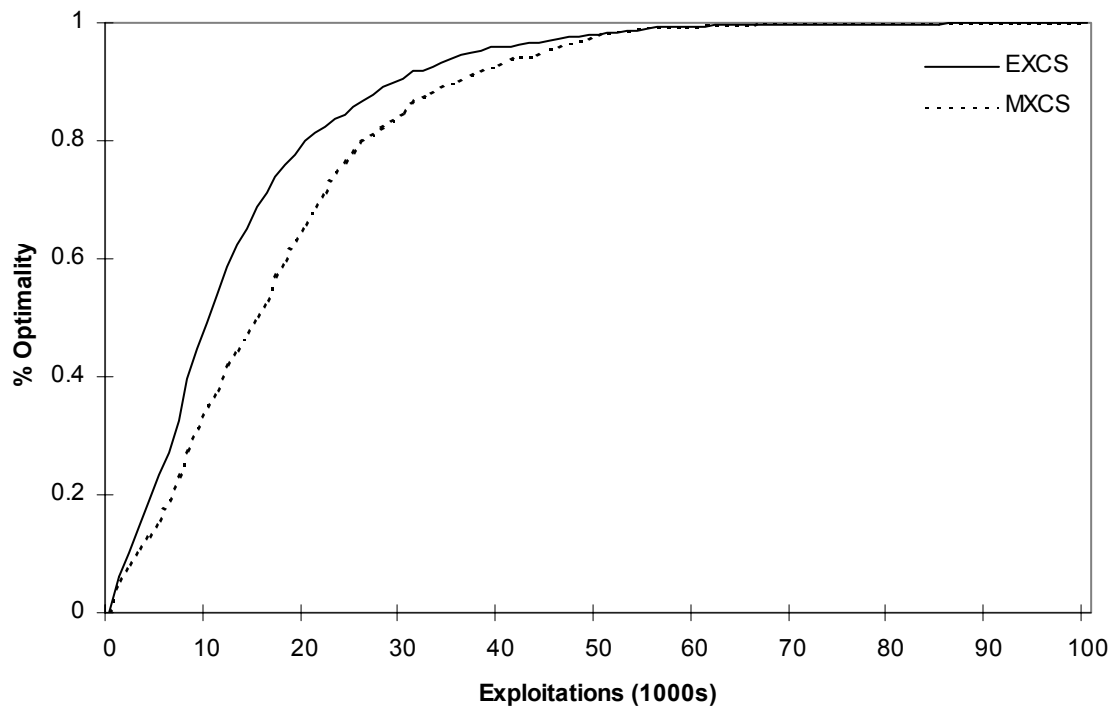


FIGURE 10. Optimality of EXCS and MXCS in 6-mux with  $N=40$  and System Parameters as in Table A-3. Curves are Averages of 100 Runs Sampled Every 600 Exploitations.

( $\approx 25k$  for EXCS compared to  $\approx 48k$  for MXCS). However, as shown in Figure 10, though EXCS has a slight initial advantage in optimality level achieved, both achieve 100% optimality near the same time (approximately 60k exploitations).

$N=20$ . In order to further test and compare the abilities of EXCS and MXCS in size-constrained systems the 6-Mux problem was investigated with the population size further reduced to  $N=20$ .

As shown in Figure 11, EXCS achieves 100% performance after approximately 37k exploitations while MXCS is incapable of settling on a successful

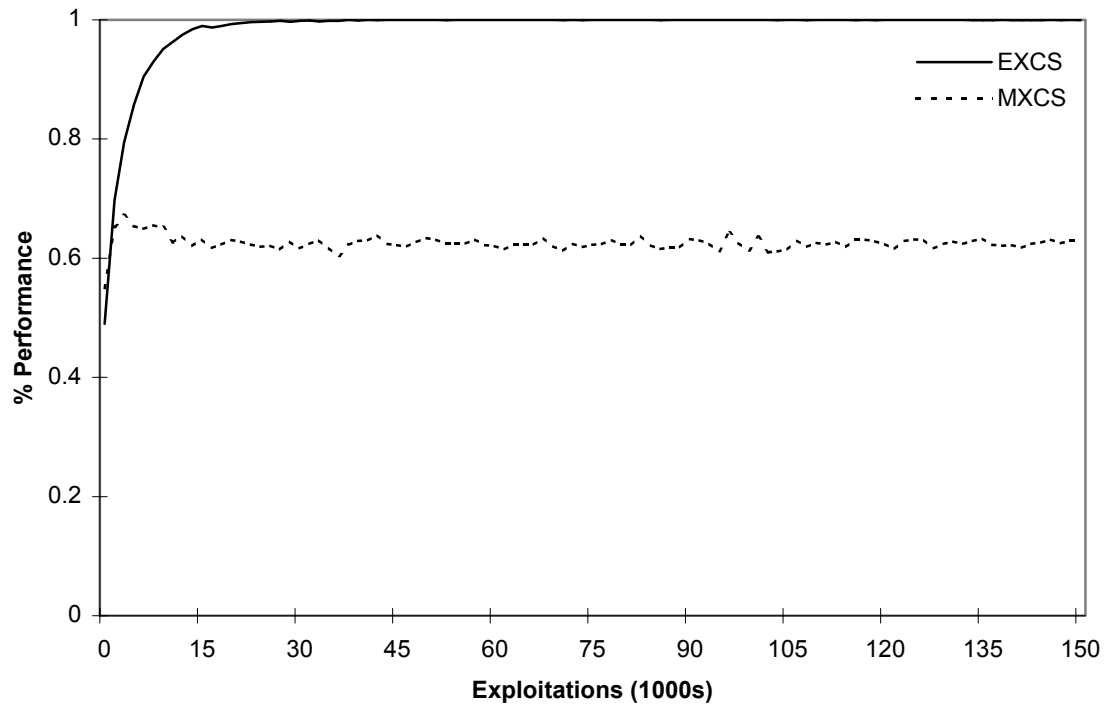


FIGURE 11. Performance of EXCS and MXCS in 6-mux with  $N=20$  and System Parameters as in Table A-3. Curves are Averages of 100 Runs Sampled Every 1,500 Exploitations.

population with the reduced population size. Note that this solution required roughly twice the number of exploitations for a population size half that of the  $N=40$  problem.

Figure 12 shows that even in this extremely size-constrained system EXCS is able to achieve the optimally effective set of classifiers after approximately 90k exploitations. These results indicate that, as anticipated, an effectiveness-based system focusing on accurate and rewarding classifiers is able to make better use of population resources in a severely size-constrained system.

While the performance demonstrated by EXCS in the 6-Mux problem is impressive, it is by no means unexpected as the 6-Mux problem is an ideal candidate for

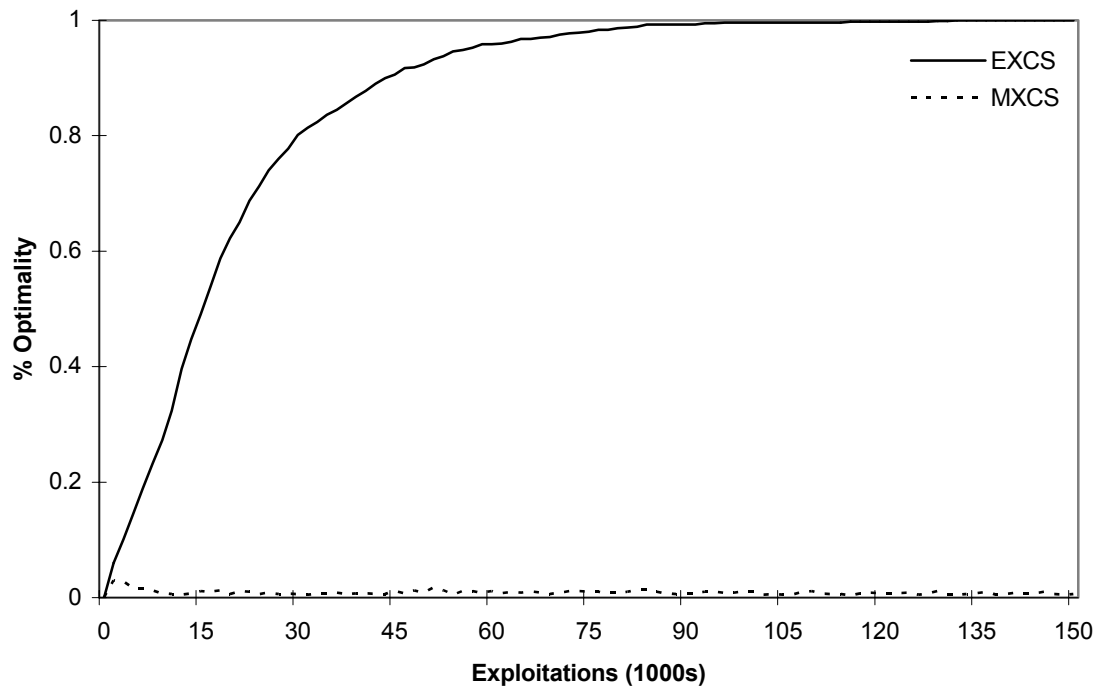


FIGURE 12. Optimality of EXCS and MXCS in 6-mux with  $N=20$  and System Parameters as in Table A-3. Curves are Averages of 100 Runs Sampled Every 1,500 Exploitations.

such a system. The unbiased reward function utilized prevents the system from suffering from the problems associated with strength-based systems, namely greedy classifier selection by the GA and the development of overgeneral classifiers.

The results do however demonstrate two points. First, that there are some problems for which strength-based (or in this case effectiveness-based) systems are well suited. Second, the performance observed in the severely size-constrained problem hints at the potential of the effectiveness-based system in resource-constrained configurations.

This potential is further exposed by the following experiments in the Woods-2 environment.

### Woods-2

While the Woods-2 environment used in the following experiments is the same as that used in Chapter V, one change was made to the exploration episode to allow a direct comparison between an accuracy-based system like the MXCS and an effectiveness-based system such as EXCS.

Complete Exploration. As EXCS focuses population resources on effective classifiers and discards the rest, it tends to develop a population containing far fewer actions per match set than XCS (*i.e.*, a best action map as previously discussed). One byproduct of this incomplete map observed in the Woods-2 environment is a significant reduction in the average number of exploration steps taken by EXCS on each exploration episode.

During an exploration episode in the Woods-2 problem, the animat's next move is selected randomly from the set of actions comprising the current match set, with the episode ending after the animat happens upon food or a maximum number of steps have been taken. Due to the fact that XCS tends to develop a complete map while EXCS tends to develop a best-action map, on average an animat in XCS takes far more steps during exploration due to the greater number of actions available to choose from. When exploring under EXCS the animat is implicitly driven towards food (and hence ending the episode) due to the limited number of actions present in each action set and their tendency to direct the animat towards the nearest food.

To allow a direct comparison between systems forming complete-maps (*e.g.*, XCS and MXCS) and those producing best-action maps (*e.g.*, EXCS), the Woods-2 exploration episode was modified to implement a *complete exploration*. Rather than starting each episode with the animat in a random location and ending upon the discovery of food or when the maximum number of steps had been taken, the exploration episode was modified to place the animat in a random location after every parameter update of the previous action set, or after updates of the immediate action set should food be reached on that step. In this way, the exploration episodes of each system (MXCS or EXCS) always take an equivalent number of steps and therefore the systems utilize approximately the same number of GA invocations.

$N=80$ . Figure 13 shows the performance of EXCS and the MXCS in Woods-2 with a population size of  $N=80$ . As shown, EXCS achieves higher performance than MXCS though the difference is relatively minor and diminishes towards the end of the run - with EXCS achieving an average of 1.70 steps to food on the last 1,000 exploitation episodes and MXCS averaging 1.92 steps during the same period. It is worth remembering at this point that XCS was unsuccessful in the Woods-2 problem at or below this size.

$N=40$ . Figure 14 shows the performance of EXCS and MXCS in Woods-2 with  $N=40$ . Again, EXCS is able to develop a higher performance population than the accuracy-based MXCS, though the difference is more pronounced at this population size.

It is interesting to note the difference between the abilities of EXCS and MXCS to scale to the smaller population used in the  $N=40$  system. Though it does take

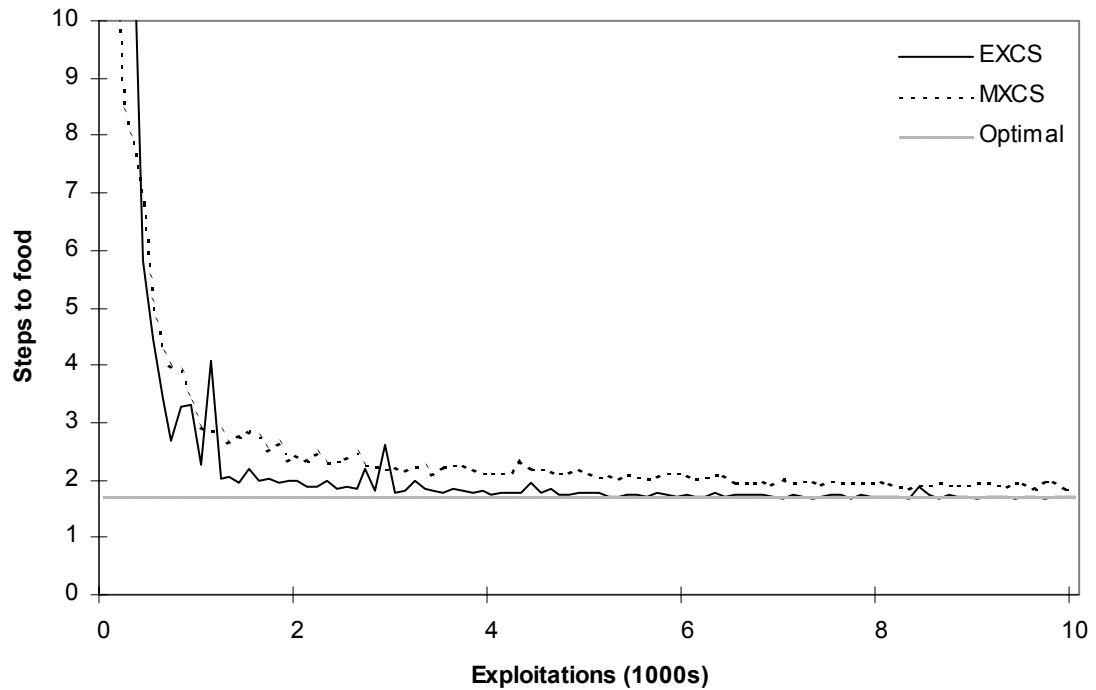


FIGURE 13. Performance of EXCS and MXCS in Woods-2 with  $N=80$  and System Parameters as in Table A-4. Curves are Averages of 10 Runs Sampled Every 100 Exploitations.

longer for EXCS to achieve a comparable level of performance in the  $N=40$  system compared to the  $N=80$  system, the final population achieves approximately the same performance, that is, 1.74 steps on average for the last 1,000 exploitations with  $N=40$  compared to 1.70 for the  $N=80$  configuration. MXCS, on the other hand, achieves an average performance level during the last 1,000 exploitation episodes of 4.25 with  $N=40$  compared to 1.92 for  $N=80$ .

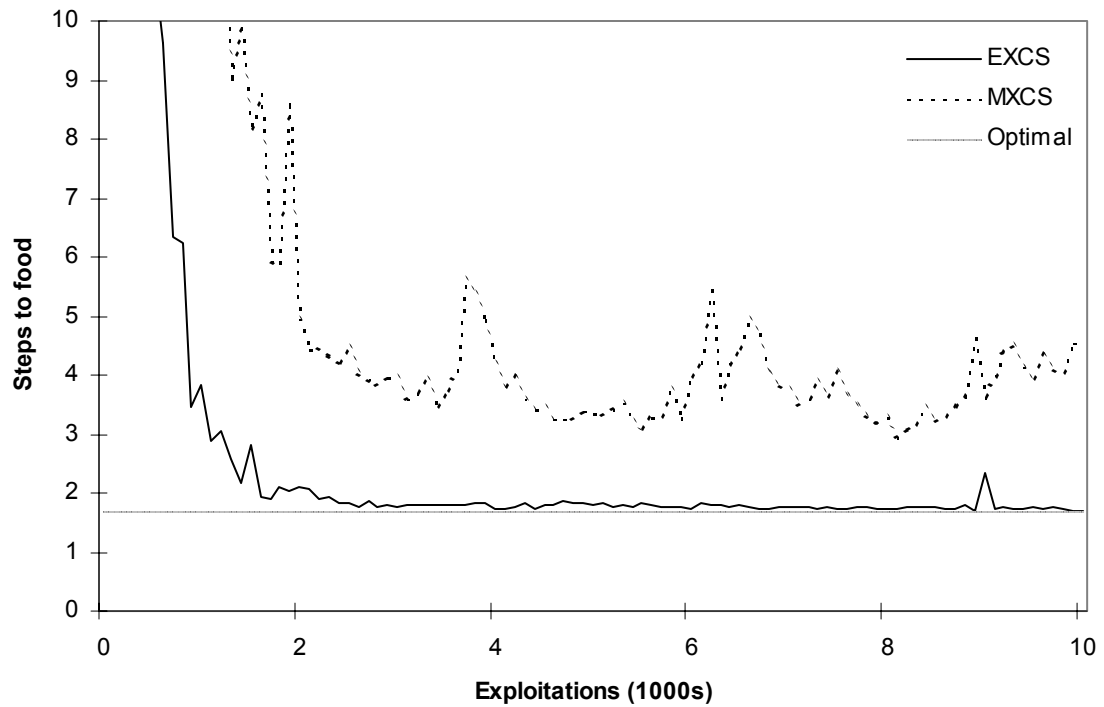


FIGURE 14. Performance of EXCS and MXCS in Woods-2 with  $N=40$  and System Parameters as in Table A-4. Curves are Averages of 10 Runs Sampled Every 100 Exploitations.

### Summary of Results

The results reported in this chapter demonstrate that the focus on effectiveness appears to allow EXCS to scale better to the more size-constrained configurations than even the modified XCS presented in the previous chapter. By developing a best-action map, EXCS is able to make better use of the system's resources while still achieving a high level of performance. Though an incomplete map of the environment is not ideal, it may be the only viable alternative for certain problem/resource configurations.

## CHAPTER VI

### CONCLUSIONS AND FUTURE

#### RESEARCH

##### Conclusions

While XCS has been shown to be able to develop complete, compact and accurate mappings of a payoff landscape, this has been accomplished in the literature through the use of significant system resources (*i.e.*, population size) relative to the size of the search space. This potentially excessive resource requirement may prevent XCS from being a viable technique for solving resource-constrained problem/platform configurations. This may prevent XCS from scaling to solve larger and more complex problems, thus making application of XCS to many real-world problems impractical.

In an effort to better understand and improve upon the range of problems to which XCS can be applied, this work focused upon the limiting case represented by size-constrained systems. By investigating the behavior of XCS in configurations of well known problems (*i.e.*, 6-Mux and Woods-2) operating under size-constraints, it was hoped that a clearer picture of the underlying mechanisms causing the resource requirements might be gained and modifications developed to overcome this limitation.

To this end, this work has presented two phases of modifications to the original XCS framework intended to allow an XCS-based system to operate effectively in size-constrained configurations. These modifications can be used to develop either a

complete or a best action map of the payoff landscape depending upon the requirements of the problem environment.

The first set of modifications, seemingly minor functional changes made to XCS, allow an accuracy-based XCS implementation (MXCS) to operate more effectively in size-constrained systems while still developing a complete map of the payoff landscape. Experimental results reported for the 6-Mux and Woods-2 problems demonstrate that the first set of proposed modifications allow a modified accuracy-based XCS to achieve high performance in size-constrained systems where the original XCS implementation is unable to develop a successful population. Our hope is that these same modifications will scale into larger and more difficult problems allowing XCS to achieve higher performance without requiring a prohibitive increase in population resources.

The second set of modifications was then introduced, comprising EXCS, a system utilizing effectiveness as the classifier fitness measure and the modifications necessary for its implementation. By measuring the effectiveness of classifiers in terms of their prediction magnitude, prediction accuracy and condition generality, relative to the match sets in which they occur, EXCS is able to focus on the development of accurate and rewarding classifiers, thus allowing the system to make better use of scarce resources. Experimental results were then presented demonstrating the improvement in performance observed when using EXCS in severely size-constrained systems.

The results presented for EXCS support the hypothesis that the use of an effectiveness-based fitness measure may be superior to the use of accuracy alone in certain problems and configurations. By focusing on developing and retaining classifiers

that are accurate as well as rewarding, EXCS combines the advantages of strength-based and accuracy-based fitness. Namely, as in a strength-based system, it recognizes the fact that where system success is measured in terms of the magnitude of the external rewards received, it makes sense to attempt to identify and favor classifiers that achieve the greatest amount of reward when the population size prevents the development of a complete map of the environment. Furthermore, it recognizes that accuracy of payoff predictions is essential in directing the GA search of the solution space and in the process of selecting the appropriate classifier for activation.

In conclusion, by combining the use of an effectiveness-based fitness measure with a niched GA, EXCS is able to overcome the key shortcomings of strength-based systems identified. The factors of accuracy and generality allow the system to develop accurate generalizations while the niched GA prevents the population from being taken over by classifiers in high payoff-garnering states (*i.e.*, greedy classifier selection). These factors combine in EXCS to produce a hybrid system that maximizes the effectiveness of the available system resources while retaining the tendency exhibited by XCS to develop accurate and maximally general rules.

### Future Research

Though the results presented in this work make a strong argument for the use of the proposed modifications in size-constrained systems, much work remains in identifying and quantifying the advantages and disadvantages of the modifications in a broader range of problems and configurations. As was demonstrated, the proposed systems operate more effectively in the presence of size-constraints by intensifying the

system's focus on fit and effective population members. However, this intensified focus comes at the cost of slowing the search of the payoff landscape, essentially trading search time (*i.e.*, iterations) for solution stability.

Therefore, some might argue that it does little good to move the prohibitive resource requirements for solving a complex problem from the size of the population to the amount of time required to reach a solution. It would seem that this argument could be responded to in two ways.

First, as the size of the population increases, the amount of real time required to perform operations such as condition checking (*i.e.*, generating the match set), conflict resolution, and deletion-vote calculations increases accordingly, with the increase impacting the system on each iteration. Thus, depending on the problem, the benefit from an increased population size may at some point be outweighed by the cost in terms of the time required for the execution of each iteration.

Second, for some problems which push the limits of the given hardware/software configuration on which the system is running it simply may not be physically possible to increase the size of the population or the amount of CPU utilization available due to physical limitations. For example, the designers of many computing devices utilizing embedded controllers (*e.g.*, networking equipment, handheld computers, "smart" appliances, etc.) must operate under severe resource constraints, with memory and CPU utilization at a premium. Therefore, it makes sense to investigate techniques that allow a problem to be solved, albeit in a greater amount of time, as moving the resource requirements into the time domain may be the only option available.

With this in mind, a more thorough investigation of the actual time required by the various XCS implementations in various configurations to achieve comparable solutions may go a long way towards helping to better understanding the usefulness and applicability of the proposed modifications.

Also of interest may be an investigation into the best approach to take in solving large or unknown problems yielding little a-priori knowledge for use in determining the appropriate settings for system parameters, such as population size and the new parameters of constraint measure,  $\Omega$ , and accuracy discount rate,  $\psi$ . While the functions presented here for determining  $\Omega$  and  $\psi$  appear to offer good performance in all problems investigated to date, the use of dynamic or self-tuning functions may provide higher performance.

For selecting the appropriate population size it may be of use to note that, as the results presented in this work indicate, it is possible to achieve some success in small populations. Therefore, it may be beneficial to employ a variable population size that begins small and increases over time. In this way the system may be able to leverage the more general classifiers developed in the size-constrained population to seed the larger population available after the next size increase. This may be beneficial as the system can then develop from more general to more specific classifiers, rather than the inverse as is typical now and which comes at a high initial cost, as a large number of specific but ineffective classifiers must be initially evaluated and discarded.

Lastly, while the effectiveness and fitness functions presented for EXCS in Chapter VI have been demonstrated to be effective in the two classes of problems

evaluated in this work (*i.e.*, 6-Mux and Woods-2), they were chosen for their simplicity and, as such, many other functions combining strength and accuracy could be utilized. One avenue of future research that may be of interest is in designing the fitness function so as to control the size of the match sets, possibly by using some other factors (such as the constraint measure and peer group estimate) to scale the degree to which the relative effectiveness of the classifiers in the set are used in calculating their fitness. In this way a fitness function could be devised which allows for less greedy classifier selection in match sets with few members, or in system configurations with little or no size-constraint. Alternatively, the fitness of a classifier could be based on an ordinal ranking of the classifier's effectiveness relative to the members of the niches in which it resides. This would yield a fitness measure that had the property of evenly distributing classifiers among match sets, even in a strength/effectiveness-based system.

## REFERENCES

## 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, pp. 245-268.
- [3] Holland J. H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975. Republished by the MIT Press, 1992.
- [4] Wilson S.W. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2), 1995, pp. 149-175.
- [5] Kovacs T. Strength or Accuracy? Fitness calculation in learning classifier systems. In: P. L. Lanzi, W. Stolzmann, S. W. Wilson (Eds.), *Learning Classifier Systems, From Foundations to Applications*, Springer-Verlag, Berlin, 2000.
- [6] Kovacs T. *A Comparison of Strength and Accuracy-Based Fitness in Learning Classifier Systems*, Ph.D. Dissertation, School of Computer Science, University of Birmingham, 2002
- [7] Cliff D., and Ross S. Adding Temporary Memory to ZCS. *Adaptive Behavior*, 3(2),1995, pp. 101-150.
- [8] Booker L.B. *Intelligent behavior as an adaptation to the task environment*, Ph.D. Dissertation, 1982, Computer and Communication Sciences, The University of Michigan.
- [9] Kovacs T. *Evolving Optimal Populations with XCS Classifier Systems*. Master's thesis, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996.
- [10] Mitchell M. *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, 1996.
- [11] Smith S. F. *A Learning System Based on Genetic Adaptive Algorithms*. Ph.D. Dissertation, University of Pittsburgh, 1980.

- [12] Goldberg D. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA, Addison-Wesley, 1989.
- [13] Fogel D.B. *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence*. Second Edition, IEEE Press, Piscataway, NJ, 2000.
- [14] Holland J. H., and Reitman J.S. Cognitive systems based on adaptive algorithms. In: D.A. Waterman and F. Hayes-Roth (Eds.), *Pattern-directed inference systems*. New York, Academic Press. 1978.
- [15] Wilson S. W. *Knowledge growth in an artificial animal*. Proceedings of the Tenth International Joint Conference on Artificial Intelligence, 217-220, Los Angeles, CA, Morgan Kaufman, 1985.
- [16] Hartley A. Accuracy-based fitness allows similar performance to humans in static and dynamic classification environments. In: W. Banzhaf et al. (Eds.), *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 266-273, Morgan Kaufmann, 1999.
- [17] Wilson S.W. ZCS: A zeroeth level classifier system, *Evolutionary Computation*, 2(1), 1994, pp. 1-18.
- [18] Watkins C. *Learning from Delayed Rewards*. Ph.D. Dissertation, 1989, Cambridge University.
- [19] Watkins C., and Dayan, P. Technical note: Q-learning. *Machine Learning*, 8, 279-292, 1992.
- [20] Butz M.V., and Wilson S.W. *An Algorithmic Description of XCS*, IlliGAL Report No 2000017, April 2000, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.
- [21] 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.
- [22] Venturini G. *Apprentissage Adaptatif et Apprentissage Supervise par Algorithme Genetique*. These de Docteur enScience (Informatique), Universite de Paris-Sud, 1994.
- [23] Sutton R. S. and Barto A. G. *Reinforcement Learning, An Introduction*. The MIT Press, Cambridge, MA, 1998.

- [24] Kovacs T. Two Views of Classifier Systems. In: P.L. Lanzi, W. Stolzmann and S.W. Wilson, (Eds.), *Advances in Learning Classifier Systems*, pages 74-87. Springer-Verlag, 2002.
- [25] 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, The MIT Press, Cambridge, MA., 1996.
- [26] Kovacs T. *Deletion Schemes for Classifier Systems*. In: W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, (Eds.), GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 329-336. Morgan Kaufmann, San Francisco (CA), 1999.
- [27] Dawson D., and Juliano B. Modifying XCS for size-constrained systems. To appear in *A Special Issue on Soft Computing in Neural Network World, International Journal on Neural and Mass-Parallel Computing and Information Systems*, 2002.
- [28] Barry A. JXCS, 1999. <http://www.cs.bath.ac.uk/~amb/LCSWEB/jxcsawt.zip> (current September 23, 2002).

## APPENDIX A

Table A-1 Parameter settings for the systems reported in Figures 2, 3, 4 and 5.

Parameter		Fig. 2,3		Fig. 4,5	
		XCS	MXCS	XCS	MXCS
Population size	$N$	400	400	40	40
GA threshold	$\theta_{ga}$	0	0	185	25
Deletion threshold	$\theta_{del}$	25	25	25	25
Subsumption threshold	$\theta_{sub}$	20	20	20	20
Covering threshold	$\theta_{mna}$	1	1	1	1
Learning rate	$\beta$	0.2	0.2	0.2	0.2
Discount factor	$\gamma$	0.71	0.71	0.71	0.71
Accuracy criterion	$\varepsilon_0$	0.01	0.01	0.01	0.01
Accuracy falloff rate	$\alpha$	0.1	0.1	0.1	0.1
Accuracy power	$\nu$	5	5	5	5
Generality rate	$P\#$	0.33	0.33	0.33	0.33
Mutation rate	$\mu$	0.04	0.04	0.04	0.04
Crossover rate	$\chi$	0.8	0.8	0.8	0.8
Deletion fitness threshold	$\delta$	0.1	0.1	0.1	0.1
Size Constraint	$\Omega$	NA	1.0	NA	6.8
Accuracy discount rate	$\psi$	.NA	NA	NA	NA
Culling threshold	$\theta_{cull}$	NA	NA	NA	NA
[A]Subsumption threshold	$\theta_{sub[A]}$	640	640	640	640

Table A-2 Parameter settings for the systems reported in Figures 7 and 8.

Parameter		Fig. 7		Fig. 8	
		XCS	MXCS	XCS	MXCS
Population size	$N$	800	800	80	80
GA threshold	$\theta_{ga}$	25	3	500	25
Deletion threshold	$\theta_{del}$	25	25	25	25
Subsumption threshold	$\theta_{sub}$	20	20	20	20
Covering threshold	$\theta_{mna}$	1	1	1	1
Learning rate	$\beta$	0.2	0.2	0.2	0.2
Discount factor	$\gamma$	0.71	0.71	0.71	0.71
Accuracy criterion	$\varepsilon_0$	0.01	0.01	0.01	0.01
Accuracy falloff rate	$\alpha$	0.1	0.1	0.1	0.1
Accuracy power	$\nu$	5	5	5	5
Generality rate	$P\#$	0.5	0.5	0.5	0.5
Mutation rate	$\mu$	0.01	0.01	0.01	0.01
Crossover rate	$\chi$	0.8	0.8	0.8	0.8
Deletion fitness threshold	$\delta$	0.1	0.1	0.1	0.1
Max exploration steps		100	100	100	100
Size Constraint	$\Omega$	NA	1.0	NA	10.7
Accuracy discount rate	$\psi$	NA	NA	NA	NA
Culling threshold	$\theta_{cull}$	NA	NA	NA	NA
[A]Subsumption threshold	$\theta_{sub[A]}$	700	700	700	700
Complete exploration		off	off	off	off

Table A-3 Parameter settings for the systems reported in Figures 9, 10, 11 and 12.

Parameter		Fig. 9,10		Fig. 11,12	
		MXCS	EXCS	MXCS	EXCS
Population size	$N$	40	40	20	20
GA threshold	$\theta_{ga}$	25	25	25	25
Deletion threshold	$\theta_{del}$	25	25	25	25
Subsumption threshold	$\theta_{sub}$	20	20	20	20
Covering threshold	$\theta_{mna}$	1	1	1	1
Learning rate	$\beta$	0.2	0.2	0.2	0.2
Discount factor	$\gamma$	0.71	0.71	0.71	0.71
Accuracy criterion	$\varepsilon_0$	0.01	0.01	0.01	0.01
Accuracy falloff rate	$\alpha$	0.1	0.1	0.1	0.1
Accuracy power	$\nu$	5	5	5	5
Generality rate	$P\#$	0.33	0.33	0.33	0.33
Mutation rate	$\mu$	0.04	0.04	0.04	0.04
Crossover rate	$\chi$	0.8	0.8	0.8	0.8
Deletion fitness threshold	$\delta$	0.1	$\infty$	0.1	$\infty$
Size Constraint	$\Omega$	6.8	6.8	10.3	10.3
Accuracy discount rate	$\psi$	NA	0.999	NA	0.999
Culling threshold	$\theta_{cull}$	NA	0.1	NA	0.1
[A]Subsumption threshold	$\theta_{sub[A]}$	640	640	640	640

Table A-4 Parameter settings for the systems reported in Figures 13 and 14.

Parameter		Fig. 13		Fig. 14	
		MXCS	EXCS	MXCS	EXCS
Population size	$N$	80	80	40	40
GA threshold	$\theta_{ga}$	25	25	25	25
Deletion threshold	$\theta_{del}$	25	25	25	25
Subsumption threshold	$\theta_{sub}$	20	20	20	20
Covering threshold	$\theta_{mna}$	1	1	1	1
Learning rate	$\beta$	0.2	0.2	0.2	0.2
Discount factor	$\gamma$	0.71	0.71	0.71	0.71
Accuracy criterion	$\varepsilon_0$	0.01	0.01	0.01	0.01
Accuracy falloff rate	$\alpha$	0.1	0.1	0.1	0.1
Accuracy power	$\nu$	5	5	5	5
Generality rate	$P\#$	0.5	0.5	0.5	0.5
Mutation rate	$\mu$	0.01	0.01	0.01	0.01
Crossover rate	$\chi$	0.8	0.8	0.8	0.8
Deletion fitness threshold	$\delta$	0.1	$\infty$	0.1	$\infty$
Max exploration steps		100	100	100	100
Size Constraint	$\Omega$	10.7	10.7	14.2	14.2
Accuracy discount rate	$\psi$	NA	0.999	NA	0.999
Culling threshold	$\theta_{cull}$	0.1	0.1	0.1	0.1
[A]Subsumption threshold	$\theta_{sub[A]}$	700	700	700	700
Complete exploration		on	on	on	on