Improving Performance in Size-Constrained Extended Classifier Systems

Devon Dawson

Hewlett-Packard, 8000 Foothills Blvd, Roseville CA 95747-5557, USA
devon.dawson@hp.com

Abstract. Extended Classifier Systems, or XCS, have been shown to be successful at developing accurate, complete and compact mappings of a problem's payoff landscape. However, the experimental results presented in the literature frequently utilize population sizes significantly larger than the size of the search space. This resource requirement may limit the range of problem/hardware combinations to which XCS can be applied. In this paper two sets of modifications are presented that are shown to improve performance in small size-constrained 6-Multiplexer and Woods-2 problems.

1 Introduction

An extended classifier system, or XCS, is a rule-based machine learning framework developed by Stewart Wilson [3,15,16] that has been shown to posses some very appealing properties. Chiefly, XCS has been found to develop accurate, complete and compact mappings of a problem's search space [6,10,15,16]. However, to be successful XCS can require a population size large enough to allow the system to maintain, at least initially, a number of unique classifiers (i.e., macroclassifiers) constituting a significant portion of the size of the search space. This potentially prohibitive resource requirement can prevent XCS from being successful where the system is *size-constrained*, that is, where the population size is smaller than the search space.

As XCS-based systems find application in industry (and particularly in cost-constrained commercial applications), size-constrained systems are likely to be frequently encountered. In an effort to broaden the range of situations in which XCS-based systems can be utilized, this paper presents two sets of modifications that have been found to significantly improve system performance in small size-constrained problem/population configurations.

Improving upon techniques employed by Dawson and Juliano in [5], the first set of mostly minor modifications attempt to make more cautious use of the limited resources available to a size-constrained system. The second modified system builds upon the first and, among other changes, uses an effectiveness-based fitness measure (i.e., a strength/accuracy hybrid) - thus resulting in a system that develops a partial map of the payoff landscape. Experimental results are reported that demonstrate the performance improvement observed when using the proposed modifications in size-constrained 6-Multiplexer and Woods-2 problems.

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 1870–1881, 2003.

2 XCS: Developing Accurate, Complete, and Compact Populations

Building upon his strength-based ZCS [14], Wilson introduced XCS in [15] as a solution to many of the problems encountered in traditional strength-based classifier systems. Namely, the problems of overgeneral classifiers, greedy classifier selection, and the interaction of the two [4,8,9,10,15]. Through the use of a simplified architecture, an accuracy-based fitness measure, a Q-Learning [12] like credit distribution algorithm and a niche Genetic Algorithm (GA) [1] – XCS is motivated towards the development of accurate, complete and compact maps of a problem's search space.

While earlier work had utilized various forms of accuracy as a factor in determining a classifier's fitness, Wilson demonstrated with XCS that the use of accuracy alone could lead the GA to develop a stable population of classifiers that accurately predict the reward anticipated as a result of their activation.

The completeness and compactness of the solution developed by XCS stems from its use of a niche GA as well as the accuracy-based fitness measure. The niche GA, introduced by Booker [1], considers classifiers to be members of specific, functionally related subsets of the population and operates upon these subsets, or niches, rather than on the population as a whole. By calculating a classifier's fitness relative to the members of the niches in which it resides and searching within these niches, the GA drives the system towards the development of the minimal number of accurate classifiers required to completely cover the search space.

3 Size-Constrained Extended Classifier Systems

Although the populations ultimately developed by XCS tend to be compact, and therefore significantly smaller than the search space (to a degree dependent upon the amount of generalization possible in the problem environment), the early stages of a run can require that the population contain a large number of classifiers, often constituting a significant percentage of the problem's search space [6,10,11,15,16]. Typically this is not a problem as the population size is set large enough to more than encompass the problem's search space. However, for practical application this may not always be possible as the size of the search space can simply exceed the physical limitations of the hardware on which the system is to execute, or the population size may be limited by the need to share system resources with other processes.

Lanzi identified in [11] that the tendency to generalize inherent in XCS may impair the system's ability to succeed in environments which afford little opportunity for generalization, and that this effect could be diminished through the use of a new *Specify* operator implemented in his XCSS. The Specify operator attempts to identify overly general classifiers and produces specialized offspring from those classifiers with some number of their don't-care bits (i.e., #'s) replaced by the corresponding bits in the current system input. Lanzi showed that as the population size was reduced, the performance of XCS and XCSS diminished in the Maze4 problem [11].

While Lanzi's work was focused on improving the performance of XCS in problems allowing little generalization, this work aims to improve performance in size-

constrained problem/population combinations. For the purposes of this paper we define a size-constrained system to be one in which the population size, N, is smaller than the size of the search space, S. In order to quantify the amount of size-constraint in a system (as required by some of the modifications proposed in this work), a new size-constraint measure, Ω, is introduced and determined by:

$$\Omega = 1 + \ln((S / N)^c)) . \quad (1)$$

For all experiments reported in this work the value of the power parameter, c, was set to c=5. For configurations where $N \geq S$, the size-constraint measure is set to 1. Note that this function was selected based simply on the fact that it resulted in values that appeared reasonable for the problems and population sizes evaluated in this work. A self tuning constraint measure (e.g., one taking into account such factors as the stability of the population) would seem to be in order.

4 Modified Extended Classifier Systems

This section presents the first set of modifications that are intended to improve the performance of size-constrained systems while still allowing the development of a complete map. For the sake of comparison the system utilizing these modifications is termed the Modified Extended Classifier System, or MXCS.

4.1 Experience-Based GA Trigger Algorithm

In XCS, the GA 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 episodes) since the last GA activation on the set exceeds the given threshold value (θ_{ga}).

While the overall impact of this triggering algorithm may be beneficial, its experience-independent nature becomes problematic in size-constrained systems where the brittle nature of the smaller populations can be easily disturbed by "over-clocking" the GA, that is, producing classifiers faster than they can be evaluated by the system. Though the GA trigger threshold can be increased to overcome this, it becomes difficult to determine the appropriate value due to its experience-independent nature and the increase may result in a significantly diminished learning rate.

MXCS modifies the trigger algorithm to use a simple experience-based threshold, where the GA is triggered when the given set contains any number of classifiers with experience since the GA last operated on it greater than the trigger threshold (θ_{ga}). While this experience-based algorithm appears to favor classifiers covering frequently encountered inputs, the favoritism is ameliorated by the following modifications and dissipates once a maximally-general classifier is discovered covering that niche, as the offspring will tend to be subsumed [16] into the generalist.

4.2 Focused Deletion-Selection

In XCS a classifier is selected for deletion from the population when the addition of a new classifier causes the population size to exceed the maximum size parameter (N).

The deletion-selection process begins by assigning each classifier a deletion-vote, which is the product of the classifier's numerosity (num_{cl}) and the learned action set size estimate (as_{cl}). The classifier's numerosity indicates the number of identical classifiers (i.e., *microclassifiers*) in the population represented by the given *macroclassifier*. The learned action set size estimate represents an average of the number of classifiers present in the action sets to which the classifier belongs. If the classifier's experience (exp_{cl}) is greater than the deletion threshold (θ_{del}) and its fitness is less than a constant fraction (δ) of the mean fitness of the population, then the deletion vote is scaled by the difference between the classifier's fitness and the mean population fitness. Therefore, this algorithm calculates the deletion vote as follows [3,7,15]:

$$vote_{cl} = num_{cl} * as_{cl} . \tag{2}$$

$$\text{IF } ((exp_{cl} > \theta_{del}) \text{ AND } (f_{cl} / num_{cl} < \delta * (\Sigma f_{pop} / \Sigma num_{pop}))) \tag{3}$$
$$vote_{cl} = vote_{cl} * (\Sigma f_{pop} / \Sigma num_{pop}) / (f_{cl} / num_{cl}) .$$

Once the deletion vote of each classifier is determined, the XCS deletion-selection algorithm typically uses *roulette-selection*, where a classifier is probabilistically selected from the population based on each classifier's deletion vote. However, in size-constrained systems roulette-selection can negatively impact system performance as its probabilistic nature often results in the deletion of effective and useful classifiers.

MXCS modifies the deletion-selection method to deterministically select the classifier with the highest deletion vote (i.e., a classifier is selected at random from the set of classifiers with the highest deletion vote). When using this *max-deletion* technique the deletion process maintains a strong downward pressure on the numerosity of classifiers, due to the deletion-vote factors of numerosity and action set size estimate.

Though not utilized in [5], the MXCS described in this work further increases the pressure on classifier numerosities as the deletion vote is modified by raising the numerosity factor by a power equivalent to the constraint measure, Ω, as follow:

$$vote_{cl} = num_{cl}^{\Omega} * as_{cl} . \tag{4}$$

4.3 Focused GA Selection

In XCS the GA probabilistically selects parents for use in the generation of new offspring from the current set (action or match depending on placement of the GA) with the selection probability based on the value of each classifier's fitness. MXCS intensifies the focus on fit classifiers by basing the selection probability on the fitness of each classifier raised to the power of the constraint measure, Ω, as follows:

$$\text{prob(cl)} = f_{cl}^{\Omega} / \Sigma f_{[A]}^{\Omega} . \tag{5}$$

4.4 Parameter Updates

In an effort to avoid favoring classifiers frequently chosen for activation, XCS limits the updating of classifier parameters (such as experience, error, fitness and predicted payoff), to exploration steps only. MXCS is modified to update a classifier's prediction and error estimates on both exploration and exploitation steps. Though not required for the system to succeed, this is done in order to take full advantage of all the information returned from the environment. It should be noted that a bias is introduced in that accurate classifiers chosen on exploitation steps will have more chances to refine their predictions and will tend to have more refined accuracy estimates.

5 Effectiveness-Based Extended Classifier System

This section builds upon MXCS and introduces a new effectiveness-based classifier system (EXCS) that further reduces the resource requirements of the system by focusing its resources on accurate and rewarding classifiers. This results in a far more compact *partial map* as the system discards classifiers with low payoff predictions.

Though the map developed by EXCS is an incomplete one, the use of a niche GA and the accuracy factor in the effectiveness measure allow the system to avoid the problems previously mentioned for purely strength-based systems [4,8,9,10,15].

5.1 Accuracy Function

In developing EXCS it was found that the standard accuracy function used in XCS was too sensitive to fluctuations in prediction error and too insensitive to the classifier's long term accuracy history to be used in determining the effectiveness of classifiers in highly competitive populations. This is particularly problematic in systems employing max-deletion due to the less forgiving nature of the algorithm.

The accuracy function used in EXCS uses two factors to calculate a classifier's accuracy. First a weighted average of the classifier's long-term accuracy is calculated. As shown in equations 6 and 7, this is accomplished by taking the ratio of the sum of the classifier's accuracy over time (k_sum_{cl}) and its experience (k_exp_{cl}) both discounted at each update by a constant fraction, ψ. Owing to the power series that develops with each update, the sensitivity of the function is determined by the value of $(1-\psi)^{-1}$ to which the series (i.e., $\Sigma\psi^n$) converges. Therefore, the weight attributed to an accuracy recorded n steps ago is ψ^n. A value of ψ=0.999 was used in all experiments reported in this paper.

Lastly, this new weighted accuracy average and the standard classifier accuracy measure are then averaged as shown in equation 8.

$$k\_sum_{cl} = (k\_sum_{cl} * \psi) + k_{cl} . \tag{6}$$

$$k\_exp_{cl} = (k\_exp_{cl} * \psi) + 1. \tag{7}$$

$$k\_ave_{cl} = ((k\_sum_{cl} / k\_exp_{cl}) + k_{cl}) / 2 . \tag{8}$$

Note that this accuracy measure was not found to improve the performance of XCS and was therefore omitted from the MXCS implementation. It may be that accuracy-based fitness functions require more drastic near-term changes in accuracy to be effective.

5.2 Effectiveness-Based Fitness Measure

Similar to the effectiveness measure utilized by Booker's GOFER-1 [2], which was a function of a classifier's "impact" (i.e., payoff), "consistency" (i.e., prediction error) and "match score" (i.e., specificity), EXCS measures classifier effectiveness as the product of a classifier's predicted payoff (p_{cl}), average accuracy (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:

$$eff_{cl} = p_{cl} * k\_ave_{cl} * gen_{cl} . \quad (9)$$

The fitness calculation in EXCS then proceeds as in XCS with two differences not found to improve the performance of MXCS. First, the GA in EXCS operates on the match set ([M]) with offspring actions determined by the crossover of the parent actions. This is done in order to cause 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 updates classifier fitness towards the ratio of a classifier's effectiveness and the maximum effectiveness found in the match set, as shown in equation 10.

$$f_{cl} = f_{cl} + \beta * (eff_{cl} / \text{MAX}(eff_{[M]}) - f_{cl}) . \quad (10)$$

The use of the maximum effectiveness in the match set prevents the fitness of each classifier from being impacted by the quantity of classifiers in the same match sets. This can be troublesome when comparing classifier fitness's across GA niches, as in the deletion-selection process.

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 and MXCS.

5.3 Conflict-Resolution

In order to accommodate the use of a partial map in EXCS, the conflict-resolution process must be modified. XCS appears to take 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.

EXCS, on the other hand, is at a disadvantage as the incomplete map developed results in inexperienced or inaccurate classifiers with higher predictions appearing favorable to the conflict-resolution process due to the lack of accurate classifiers in the same set (e.g., the fitness weighted prediction of a classifier that is the only member of an action set is independent of the classifier's fitness, and therefore accuracy).

This problem is overcome in EXCS by scaling the contribution of an inexperienced classifier (i.e., $exp_{cl} < \Omega * \theta_{del}$) to the calculation of the prediction and fitness sums for an action set. As shown in equations 11 and 12 this is accomplished by scaling the prediction of an inexperienced classifier by the square of a small fraction (i.e., $.01^2$), and scaling the inexperienced classifier's contribution to the fitness sum by the same small fraction (i.e., .01).

$$\Sigma (p*f)_{[A]} = \Sigma (p*f)_{[A]} + .01^2 * p_{cl} * f_{cl} . \quad (11)$$

$$\Sigma f_{[A]} = \Sigma f_{[A]} + .01 * f_{cl} . \quad (12)$$

The use of the square of the fraction in the calculation of the payoff prediction sum helps to prevent the inexperienced classifier from overestimating its predicted payoff as well as protects against situations where the classifier is the only member of the action set. Using the fraction of the classifier's fitness in the fitness sum for the action set prevents the inexperienced classifier from distorting the payoff predicted for that action set. Note that it was found to be beneficial to performance to use these scaling factors, rather than simply disregarding inexperienced classifiers.

5.4 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 before each invocation of the GA when the classifier's experience is greater than the deletion threshold ($exp_{cl} > \theta_{del}$), its effectiveness is below the minimum error parameter (ε_0) and the percentage of the population composed of such ineffective classifiers is above a new *cullling threshold* (θ_{cull} – in all experiments reported in this paper, θ_{cull}=0.1). The process employed was to first determine the set of ineffective classifiers then randomly remove classifiers from this set until the threshold is met.

The culling threshold 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 immediately 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 levels.

6 Experimental Comparisons

This section presents experimental results that compare the performance of XCS, MXCS and EXCS in size-constrained 6-Multiplexer and Woods-2 problems.

Except for changes required for the described modifications the systems were implemented to follow the XCS definition described by Butz and Wilson in [3]. System parameters for all experiments reported in this work are as in [15] except where noted. The covering threshold parameter (θ_{mna}) was set to require one classifier per match set.

This value yields better performance in size-constrained systems. For EXCS, the deletion-vote fitness fraction (δ) was disregarded (i.e., set to infinity) as this was found to improve performance. This change had an adverse effect when used in XCS or MXCS, possibly owing to the greater amount of information encoded in the effectiveness-based fitness measure. Classifier parameters were updated in the order: experience, error, prediction, and then fitness.

Subsumption deletion [3,16] was used in all experiments, though a new subsumption experience threshold ($\theta_{sub[A]}$) was introduced for use in triggering action set subsumption. Owing to the fact that the negative impact of an erroneous action set subsumption can be far more severe than an erroneous 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 be ten times the size of the problem's input space (i.e., the number of possible inputs). This gives occasionally erroneous classifiers more opportunities to be identified and removed from the system before subsuming other more accurate classifiers. Note that this level is certainly not intended to work for other problems with potentially much larger input spaces. It is merely intended to allow subsumption deletion to be used in the relatively small environments reported here.

6.1 6-Multiplexer

The multiplexer class of problems are frequently used in the literature 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 six bits in any given input string of a 6-Mulitplexer (6-Mux) are important, the two address bits and the bit at the position pointed to by the address bits (i.e., the correct output). Due to this property, the system can develop general rules with don't-care symbols in the ignored bit positions of their conditions that will accurately encode the entire set of classifiers masked by that condition. For example, while the 6-Mux problem requires 128 unique and fully defined classifiers to be complete, a smaller set of 16 *optimal* classifiers can accurately cover the entire input space given a two-level payoff landscape.

For all 6-Mux 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.

The results presented in this work for the 6-Mux problem use two different performance measures commonly reported in the literature: performance and optimality. *Performance* is a moving average of the percentage of the last 50 exploit steps that returned the correct output. *Optimality* measures the percentage of the optimal set of classifiers present in the population.

As stated, for the 6-Mux problem with two payoff levels the optimal set consists of 16 classifiers. However, in order to compare systems developing complete maps (i.e., XCS and MXCS) and partial maps (i.e., EXCS) the optimal set is restricted to those optimal classifiers that earn the maximum reward. Therefore, for the 6-mux problem the *optimally-effective* set of classifiers consists of 8 unique classifiers.

Furthermore, to facilitate a direct comparison of systems utilizing different GA triggering algorithms the GA trigger threshold (θ_{ga}) was selected for XCS so as to cause the GA to be invoked at the same rate as MXCS and EXCS. Therefore, θ_{ga} for XCS with N=40 was set to θ_{ga}=185 and θ_{ga} for XCS with N=20 was set to θ_{ga}=265. These values resulted in approximately the same number of GA invocations on average per run as the value of θ_{ga}=25 used in the experience-based MXCS and EXCS GA trigger algorithms. Note that the performance of XCS was not found to be greatly affected by any of the numerous other trigger values tested and results obtained with N=400 (not shown) closely approximated those reported by Wilson and others [6,10,15].

Performance. Figure 1 shows performance in the 6-Mux problem at the two size-constrained configurations of N=40 and N=20 (a size of N=400 is commonly used for the 6-Mux problem). As shown, XCS is unsuccessful at either constrained size. MXCS, on the other hand, achieves a successful population with N=40 but fails with N=20. EXCS succeeds at both N=40 and N=20.

Optimality. Figure 2 shows the optimality of the three systems. Due to the small population sizes XCS is unable to maintain much of the optimally-effective set of classifiers while MXCS succeeds with N=40 but performs poorly with N=20. EXCS performs well with both N=40 and N=20.

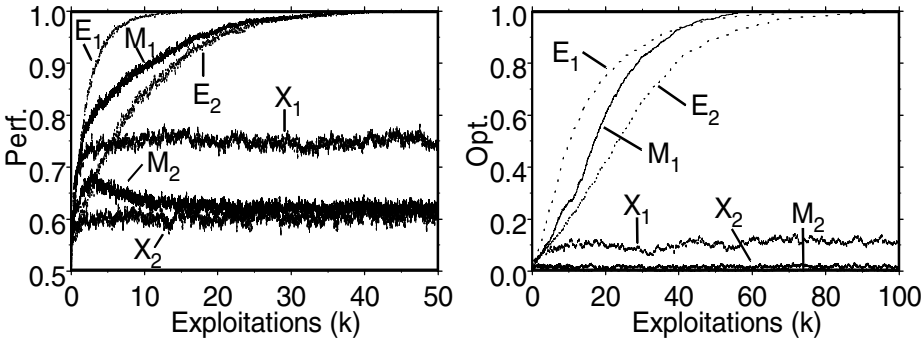

Fig. 1 (left) and Fig. 2 (right). 6-Mux performance (left) and optimality (right) of: XCS with N=40 (X_1) and N=20 (X_2), MXCS with N=40 (M_1) and N=20 (M_2), and EXCS with N=40 (E_1) and N=20 (E_2). Curves are averages of 100 runs

6.2 Woods-2

This section compares the performance of XCS, MXCS and EXCS in the Woods-2 problem. Though, as Kovacs explains in [10], Woods-2 is not a very difficult test of the abilities of a system to learn sequential decision tasks, it serves as a standard benchmark by which we can compare the system's responses to population pressure.

Woods-2, developed by Wilson [15], is a multi-step Markov problem with delayed rewards in which the system represents an "animat" [13], or organism, searching for food on a 30 by 15 grid. 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 two rows of empty space and the grid "wraps around" (e.g., moving off the edge of the grid places the animat on the opposite edge of the grid), thereby creating a continuous environment.

The system input is a 24-bit binary string representing the eight space types surrounding the location currently occupied by the animat (i.e., the 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 in response to its currently sensed surroundings. 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 systems.

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 [10,15].

As with the 6-Mux problem, the GA trigger threshold was selected for XCS in Woods-2 in order to cause the system to invoke the GA approximately the same number of times per run as MXCS and EXCS. Therefore, values of θ_{ga}=500 and θ_{ga}=1000 were used for XCS at N=80 and N=40 respectively; while a value of θ_{ga}=25 was used for MXCS and EXCS. As with the 6-Mux experiments, a variety of threshold values were evaluated and not found to significantly alter the performance of XCS at these population sizes.

One further change was made to the implementation of the Woods-2 problem in order to allow a comparison of systems developing complete and partial maps. The exploration phase of the Woods-2 problem involves placing the animat at a random location and ending the exploration once the random selection of actions causes the animat to move onto a food space (and thus garner a reward) or when the maximum number of steps have been taken. However, it was found that a system developing an incomplete map (i.e., EXCS) will tend to take far fewer steps on average during exploration as the majority of actions present will direct the animat towards food. This greatly reduces the number of steps taken during exploration and thus reduces the amount of "learning" taking place during each exploration episode.

In an attempt to level the playing field, the exploration episodes of the Woods-2 problems reported in this work utilize a *complete-exploration* technique. Under complete-exploration the animat is placed in a new random location after each update of the prior action set or when food is reached. This continues until the maximum number of steps have been taken, thereby ensuring that each system takes the same number of exploration steps per run. Note that this technique is not required for the system to learn, it is used to allow a direct comparison between the two types of systems.

Performance. Figure 3 shows the performance of the three systems in the Woods-2 problem at two size-constrained configurations of N=80 and N=40 (note the logarithmic vertical axis). As shown, the size constraint causes XCS to be unsuccessful at either population size, actually performing worse than would be expected from a random walk. MXCS achieves an average performance level of 1.94 steps by the end of

the run with N=80 and 3.83 steps with N=40. EXCS reaches average performance levels of 1.74 steps and 1.84 steps for N=80 and N=40 respectively.

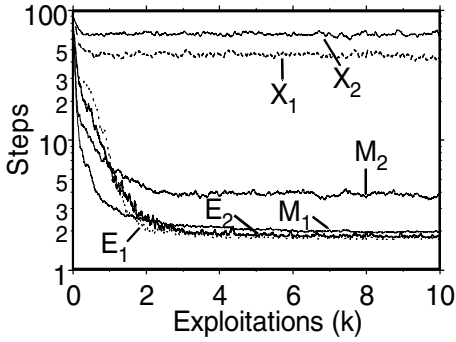

Fig. 3. Woods2 Performance of: XCS with N=80 (X_1) and N=40 (X_2), MXCS with N=80 (M_1) and N=40 (M_2), EXCS with N=80 (E_1) and N=40 (E_2). Curves are averages of 100 runs

7 Conclusions and Future Directions

This paper has attempted to broaden the range of applications to which the XCS framework can be applied by introducing modifications that may help XCS to perform in size-constrained systems. Though the solutions may take longer to reach, it makes sense to investigate alternatives that allow the system's requirements to be shifted from physical resources (i.e., population size) to system iterations as the physical resources may be limited by forces out of the implementers control.

As was demonstrated, the first modified system, MXCS, was able to outperform XCS in small size-constrained configurations by, in essence, tuning the system's algorithms to make more cautious use of the limited resources available. The second system introduced, EXCS, was shown to outperform both XCS and MXCS by taking advantage of the lower resource requirements of a partial map. Though a system developing an incomplete map such as EXCS may not be suitable for many problems, the advantages in terms of reduced resource requirements and potentially shorter time-to-solution make it appealing for those problems that allow an incomplete map.

There are a variety or areas related to this work that could be of interest for future investigation. First, a systematic evaluation of each modification is called for. Also, a more thorough investigation into the resource requirements of XCS and the relationship between population size and the ability of the system to generalize is called for. Furthermore, a comparison of the systems presented here, as well as others (such as Kovacs' SBXCS [9,10] and Lanzi's XCSS [11]), in a wider range of problems and varying size-constraint levels seems to be in order. Lastly, a comparison of these systems in terms of time-to-solution for a variety of problems and configurations may help to make an argument for the use of smaller populations, as the computational cost of an increased population size will at some point outweigh the benefit.

The author would like to thank the Hewlett-Packard Company for its support and the reviewers of this paper for their time and valuable feedback.

References

1. Booker L.B. *Intelligent behavior as an adaptation to the task environment*. Ph.D. Dissertation, The University of Michigan, 1982.
2. Booker L.B. Triggered rule discovery in classifier systems. In: J.D. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms (ICGA89)*, pages 265–274. Morgan Kaufmann, 1989.
3. Butz M.V. and Wilson S.W. An Algorithmic Description of XCS. In: Lanzi P.L., Stolzmann W. and Wilson S.W. (Eds.), *Advances in Learning Classifier Systems, LNAI 1996*, pages 253–272. Springer-Verlag, Berlin, 2001.
4. Cliff D. and Ross S., Adding Temporary Memory to ZCS. *Adaptive Behavior*, 3(2):101–150, 1995.
5. Dawson D. and Juliano B. Modifying XCS for size-constrained systems. *Neural Network World, International Journal on Neural and Mass-Parallel Computing and Information Systems, Special Issue on Soft Computing Systems*, 12(6), pages 519–531, 2002.
6. Kovacs T. *Evolving Optimal Populations with XCS Classifier Systems.* Master's Thesis, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996.
7. 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, pages 329–336. Morgan Kaufmann, San Francisco (CA), 1999.
8. 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.
9 Kovacs T. Two Views of Classifier Systems. In: Lanzi, P.L., Stolzmann W., and Wilson, S.W., (Eds.), *Advances in Learning Classifier Systems*, pages 74–87. Springer-Verlag, Berlin, 2002.
10. 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.
11. Lanzi P.L. An Analysis of Generalization in the XCS Classifier System. *Evolutionary Computation*, 7(2):125–149. 1999.
12. Watkins C. *Learning from Delayed Rewards*. Ph.D. Dissertation, Cambridge University, 1989.
13. Wilson S.W. *Knowledge growth in an artificial animal*. Proceedings of the Tenth International Joint Conference on Artificial Intelligence. pages 217–220. Los Angeles (CA), Morgan Kaufman, 1985.
14. Wilson S.W. ZCS: A zeroeth level classifier system, *Evolutionary Computation*, 2(1), pages 1–18. 1994
15. Wilson S.W. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2), pages 149–175. 1995,
16. 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*, pages 665–674. Morgan Kaufmann, San Francisco (CA), 1996.