Deletion Schemes for Classifier Systems

Tim Kovacs
School of Computer Science
University of Birmingham
Birmingham B15 2TT United Kingdom
Email: T.Kovacs@cs.bham.ac.uk
Telephone: (44) 121 414 4773

Abstract

The issue of deletion schemes for classifier systems has received little attention. In a standard genetic algorithm a chromosome can be evaluated (assigned a reasonable fitness) immediately. In classifier systems, however, a chromosome can only be fully evaluated after many interactions with the environment, since a chromosome may generalise over many environmental states. A new technique which protects poorly evaluated chromosomes outperforms both techniques from (Wilson, 1995) in two very different single step problems. Results indicate the XCS classifier system is able to learn single step problems for which no (or few) useful generalisations can be made over the input string, despite its drive towards accurate generalisation.

1 Introduction

In genetic algorithms (GA) research the issue of selecting individuals to delete from a population has received little attention compared to the issue of selecting individuals for reproduction. Further, GA deletion techniques have been incorporated into GA-based Learning Classifier Systems (LCS) without regard for the differences between the two. This paper studies existing deletion techniques and introduces one specifically tailored for LCS in an attempt to optimise their performance.

A classifier system learns if-then rules (classifiers) incrementally through interaction with a problem environment. Interaction consists of cycles of: input to the LCS, reaction by the LCS, and payoff (reward) from the environment. LCS typically use a GA for rule discovery. GAs are search mechanisms inspired by natural selection in biological species which operate by evolving successively better generations of individuals. In a standard function optimisation GA individuals are codings of parameter values, while in a classifier system's GA individuals are classifiers. In a GA various search operators (typically crossover and mutation) are used to obtain offspring from parents. In order to obtain a population of desirable individuals those judged fitter (better) are given preference when selecting parents.

XCS is a type of classifier system introduced in [1] in which the fitness of a classifier is based on the accuracy of its payoff prediction and not on the prediction itself. As a result, XCS, unlike standard LCS, tends to learn complete and accurate representations of the input/action/payoff function. In [2] I have shown that XCS can in fact learn minimal, complete and accurate representations (populations) for boolean functions. I call a population with these three characteristics an *optimal population* or [O] in my notation.[1] The reader is referred to [1] for a description of XCS, and to [4] for a study of now-standard modifications to the system. Questions and answers on XCS (and other papers on complex adaptive system) are available via NetQ at http://prediction-dynamics.com/

Another feature of XCS is the use of *macroclassifiers* [1], classifiers with a *numerosity* parameter which indicates the number of virtual copies of the classifier considered to be in the population. The numerosity of a macroclassifier is initially 1 but rises if the GA produces additional copies of the classifier, and falls if they are deleted. A macroclassifier is always treated by the system as the equivalent number of (micro)classifiers. Macroclassifiers provide interesting statistics and al-

[1] [O] for the specific case of the 6 multiplexer was first discussed by Wilson [3] who referred to it as [S6] – the solution set for the 6 multiplexer.

low a considerable increase in run-time speed as additional copies of a classifier impose no further computational load on the system. We could limit each classifier to a single or few copies and obtain increased search rates, but this would be offset by increased run times.[2] In this document graphs of population size show the number of macroclassifiers, as the size in *micro*classifiers remains at the fixed population size limit. For simplicity, macroclassifiers will be referred to simply as classifiers.

Two reproductive schemes have been used with the GA: *generational* and *steady state* reproduction. In generational reproduction the entire population is replaced at once,[3] while in steady state reproduction only a few individuals are replaced on each reproductive event. Generational reproduction is perhaps the more widely studied approach, but steady state GAs are also commonly used and are associated with classifier systems. See [5] (section C2.7) for comparison of generational and steady state systems.

1.1 Deletion Schemes

Using a fixed population size requires that each time we insert a new rule into the population we remove an old rule. Thus, under generational reproduction we need to select parents, but with steady state reproduction we need to select both parents and rules to be replaced. Since there are two points at which selective pressure may be applied in the steady state GA more flexible schemes than are normally used are possible. For example, in [6] the use of a deletion scheme with selective bias against unfit individuals removes the burden of maintaining fit elements of the population from the crossover operator. This allows the use of more disruptive crossover operators, which were found to yield better performance.

Many parent selection schemes (e.g. linear ranking, roulette wheel, tournament selection) have been used and studied (see for example [7, 8]) and all could be applied to selection of rules to be replaced. Little work, however, has specifically addressed the issue of selecting rules for replacement ([9] is an exception). A simple scheme, which will be used as a benchmark in this paper, is random selection of rules to be deleted. This scheme has no bias at all. In practice, deletion schemes which are biased against low-fitness rules are most often used. For example we might always delete the least fit rule, or we might use some stochastic scheme with a bias against less fit rules. In addition to the normal bias against low fitness rules, we can introduce other biases to help optimise system performance. One of the aims of this work is to demonstrate a successful combination of two biases in a deletion scheme.

Wilson [1] proposed two deletion techniques for XCS:

> "1. Every classifier keeps an estimate of the size of the match sets[4] in which it occurs. The estimate is updated every time the classifier takes part in an [M], using the MAM technique[5] with rate β. A classifier's deletion probability is set proportional to the match set size estimate,[6] which tends to make all match sets have about the same size, so that classifier resources are allocated more or less equally to all niches (match sets). This deletion technique is similar to one introduced in [10] for the same purpose.
>
> 2. A classifier's deletion probability is as in (1), except if its fitness is less than a small fraction δ of the population mean fitness. Then the probability from (1) is multiplied by the mean fitness divided by the classifier's fitness." (Wilson, 1995)

We will refer to these techniques as t1 and t2 respectively. The first technique is biased against classifiers with large action set sizes (i.e. classifiers which match inputs matched by many other classifiers) in order to help regulate the distribution of classifiers between action sets. Notice that there is no bias at all against unfit classifiers. The second technique is a modification of the first in which a bias against unfit classifiers is added. These classifiers have their deletion probability steeply increased in inverse proportion to their fitness. We can identify three desirable biases for deletion schemes in XCS: i) against low fitness classifiers, since they contribute least (or are detrimental) to performance ii) against classifiers which are subsumed by more general and equally accurate classifiers, since

[2] This is supported by findings (not shown) on the 6 multiplexer test, which is to be introduced shortly.

[3] The best few members may be retained in the next generation. This is called elitism. Also, if crossover is not always applied clones may appear in the next generation.

[4] The *match set* [M] is the set of classifiers which match the current input.

[5] The MAM technique is a modification of the delta rule for reinforcement learning. Under the delta rule, an estimated value approaches observed values by minimising the difference between the two. The MAM technique improves the learning rate by allowing for larger changes in the estimated value early on, somewhat like simulated annealing.

[6] Wilson has since moved the GA to the *action set* [A] (the subset of the match set which advocates the action chosen by the system). Consequently, the deletion probability is actually proportional to the size of the action set.

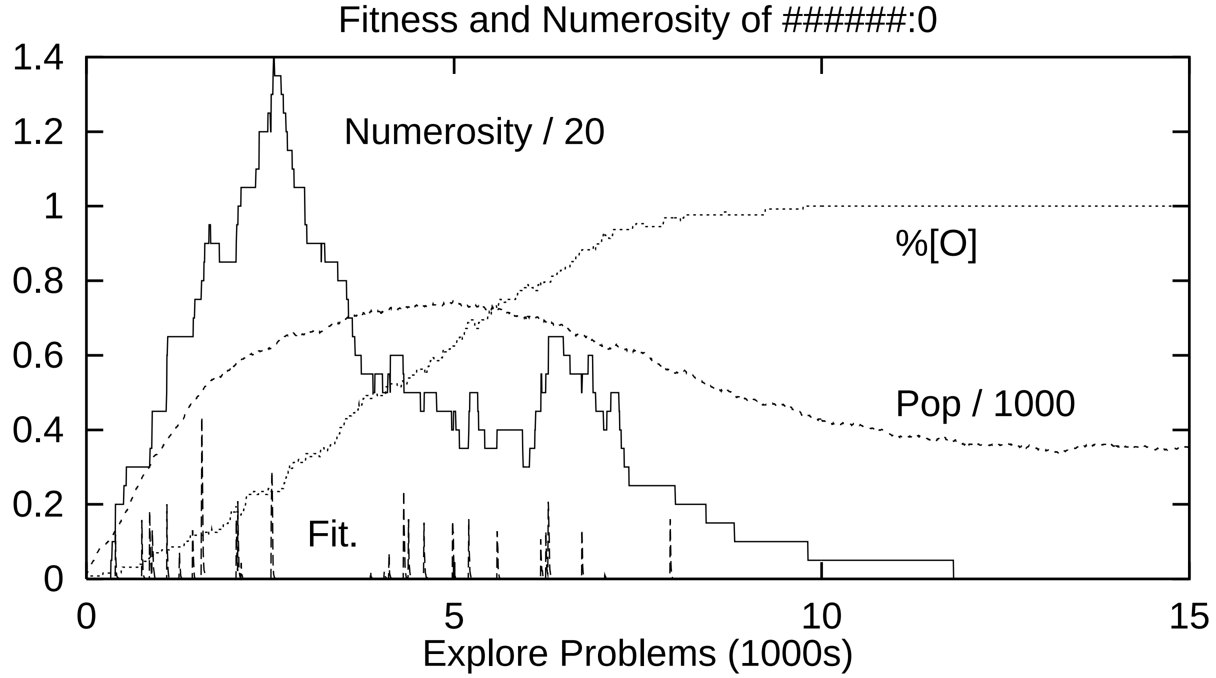

Figure 1: A single run of the 6 parity problem showing the high numerosity and low, spiky fitness of classifier ######:0 using t1.

they only drain processing power, and iii) against large action sets, in order to balance classifier allocation between action sets. Only the first two will be addressed here.

XCS will be evaluated on two two rather different Boolean functions. Both are single step problems: each input to the system is independent of all others (in fact, input strings are generated randomly). A trial consists of a single input. System settings are in appendix A.

In [1] the progress of the system in learning a problem was monitored using a statistic called performance, defined as the proportion of the last 50 inputs to which the system has responded correctly. In this work I use instead the percent of the optimal population %[O] which is present in the general population [P]. This measure is more sensitive to the progress of evolutionary search than performance: in some cases two deletion techniques have similar performance curves, but rather different %[O] curves.

1.2 The Multiplexer Function

The multiplexer function is defined for strings of length $L = k + 2^k$ where k is an integer > 0. In the 6 multiplexer problem ($k = 2$), the input to the system consists of a string of six binary digits, of which the first two (the address) represent an index (in binary) into the remaining bits (the data), and the value of the function is the value of the indexed data bit. E.g., the value of 101101 is 0 as the first two bits 10 represent the index 2 (in base ten) which is the third bit following the address. Similarly, the value of 001000 is 1 as the 0th bit after the address is indexed. The optimal population for this problem consists of the 8 classifier conditions of the form: 11###1 (i.e. where all digits in the data part are # apart from the indexed one). Unlike other classifier systems, in XCS each possible action in a state is advocated by at least one classifier. So [O] contains two classifiers for each optimal classifier condition; one for each of the two possible actions. This results in a total of 16 classifiers in [O].

Single step tests of XCS have to date been largely restricted to the multiplexer series (specifically, the 6, 11 and 20 multiplexers). Multiplexer problems are of interest as they are highly non-linear (and thus may be difficult to learn), and yet afford many useful generalisations, allowing us to test XCS's generalisation ability. However, it is important to evaluate XCS on rather different problems in order to gain a more rounded understanding of its performance.

1.3 The Even Parity Function

The even parity function returns a value of 1 if an even number of bits in the input string are 1s, otherwise it returns a 0. Using the standard ternary alphabet of classifier conditions no useful generalisations can be made for the parity function (i.e. a # in any position will result in the classifier being overgeneral). This means that the optimal population for this problem consists of one classifier for each possible input/action pair, or $2^6 = 64$ inputs $\times$ 2 actions = 128 classifiers. The parity function is at one extreme in terms of the useful generalisations which can be made. Since one of the strengths of XCS is its ability to exploit useful generalisations it will be of interest to evaluate its performance when there are none.

2 Evaluating t1 and t2

I have used t1 on the 6 multiplexer function in [11, 2] with satisfactory results. However, examination of evolved populations revealed that unfit classifiers (i.e. those with a fitness of or close to 0.0) often had considerable numerosity. Further, this problem is much worse in the case of the 6 parity problem. We can attribute this to the lack of bias against such classifiers under t1.

Although unfit classifiers are unlikely to be selected as parents, reproduction among other classifiers may produce additional copies of unfit classifiers as a result of crossover and mutation. Unfit offspring from different niches are called *lethals* (see [12] section 11), and their creation is not prevented by the use of restricted mating.[7] The extent to which this occurs will vary from problem to problem. In some cases, two high fitness strings will often cross to produce copies of a particular unfit string, e.g. 0#1 and #01 will often produce ##1 as a result of crossover. If 0#1 and #01 are fit they will tend to reproduce frequently, which will periodically increment the numerosity of ##1 without regard for its fitness.

Another possible cause of high numerosity in unfit classifiers is the presence of short periods during which these classifiers do have some fitness (fitness spikes), and during which they may participate in reproduction. Fitness spikes are visible in figure 1 which shows the numerosity and fitness of ######:0 on a single run of the 6 parity problem. This classifier matches all inputs and advocates the wrong action 50% of the time.[8] Despite the fact that its fitness is essentially 0 most of the time, it accumulates a peak numerosity of 28 (although mean peak fitness over 10 runs was only 16). Its fitness curve has spikes (one of which reaches approximately 0.45 out of a maximum of 1.0) which appear on the graph to counteract numerosity loss due to the deletion mechanism (i.e. numerosity appears to fall mainly when spikes are absent). This may not, however, be a reliable effect.

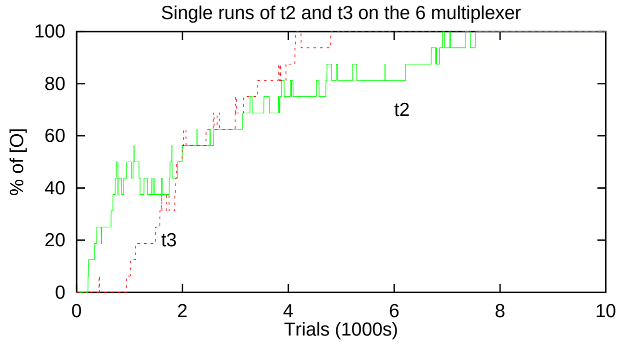

Figure 2: Using t2 elements of [O] are often discovered but then quickly discarded resulting in spikes and a longer time to convergence. In contrast, t3 rises much more steadily and converges much sooner.

Evaluation of t2 on the multiplexer problem found that it reduced population size considerably compared to t1, but that XCS's ability to respond correctly to input strings was noticeably impaired. Further, t2 is highly detrimental to evolving optimal populations. Figure 2 demonstrates the difficultly that members of [O] have in becoming established under t2. They are often discovered but quickly deleted resulting in a spiky %[O] curve and a failure to converge completely.

As figure 3 shows, t2 is very effective at eliminating low fitness classifiers. I hypothesise that it is in fact too effective in doing this, and that it removes accurate classifiers before they have a chance to gain fitness. (It may also remove inaccurate classifiers before they have a chance to contribute beneficially to the genetic search.) Part of the problem is that as accurate (and thus high fitness) classifiers are found the mean fitness of the population rises. This makes it increasingly difficult for new classifiers to become established as they are more likely to fall under the δ threshold for low-fitness penalisation (see 1.1). So the search for new classifiers becomes more inhibited as the system performs better.

[7] In XCS only classifiers from the same action set can be crossed as parents.

[8] This classifier may be interpreted as: if the input matches ###### then perform action 0. #s are wildcards so this classifier matches any input string.

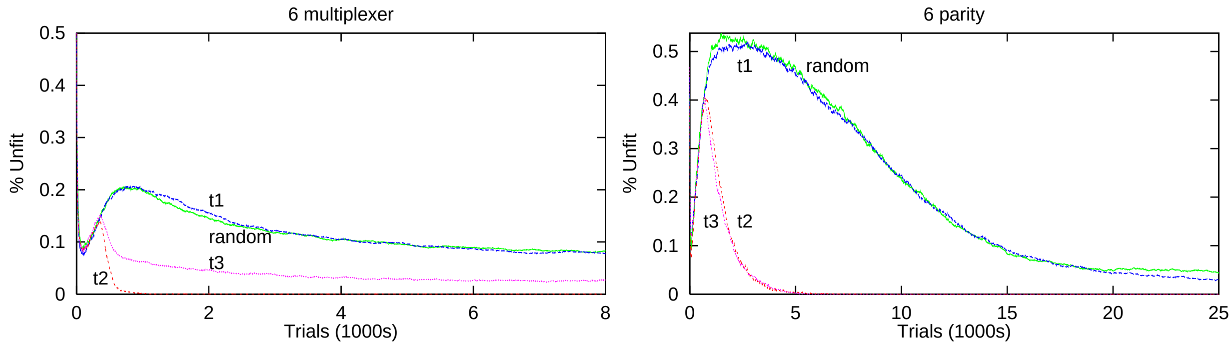

Figure 3: The % of total numerosity in the population held by classifiers with fitness below 0.001 (maximum fitness = 1). Many more unfit classifiers are present in the parity problem. The multiplexer tests were averaged over 100 runs, parity over 30.

3 Introducing t3: a technique which protects new classifiers

Because a new classifier must normally be tested on many trials before we can be certain of its fitness, we initially set its fitness to a low default value and increase it slightly each time it proves itself useful (see [1] for details).[9] This way good (i.e. accurate) classifiers gradually increase their chances of participating in reproduction as we become more confident of their utility. Bad classifiers tend not to increase in fitness and so tend not to participate in reproduction. If we were to set the initial fitness to a high value good classifiers could reproduce immediately, but then so could bad ones.

But since all classifiers initially have a low fitness, a bias against low fitness classifiers is also a bias against new classifiers, both good and bad. The stronger the bias, the more the system will tend to delete useful new classifiers before it realises how good they are. t2 suffers greatly from this problem. As a remedy, I developed a new technique called t3, which is a hybrid in which t1 is used until a classifier has been used on n trials, after which t2 is used. n is called the delay for t3 as it controls how long we delay the application of the low fitness penalty.

On both tests t3 has the fastest convergence rate, the second lowest population size and requires the least computational effort to find the optimal population (figure 4). This supports the hypothesis that protecting new classifiers is beneficial. Using t3 very little of the total numerosity in the population is held by unfit classifiers (figure 3).

[9] Fitness is initially set to 10% of the population mean.

3.1 Setting the delay parameter for t3

Figures 5 and 6 compare random, t1 and t2 to t3 with various delays. Figure 5 shows the %[O] regret, defined as 1-%[O] averaged over all trials. (This corresponds to the mean distance between the top of the graph and the %[O] curve in other figures.) Perfect performance would produce a %[O] regret of 0, while the worst possible performance would produce a regret of 1.

Recall that, although they have the same state space size, the parity problem affords no generalisations while the multiplexer affords many. The parity problem is one bounding case, while the multiplexer is close to the other. The amount of generalisation possible is an important dimension of problem difficulty for classifier systems: for one thing, a non-generalising learner is likely be more efficient when few generalisations are possible. A consequence of this dimension in this study was that a larger population size and more trials were required to learn the parity problem. Nonetheless, that similar t3 delay values yielded best results for these two very different problems suggests that the delay parameter is not very sensitive to the amount of generalisation possible. A possible reason that the parity problem requires less delay than the multiplexer problem is that classifiers in the former tend to be much more specific. Specific classifiers are updated less frequently than general ones, and so are protected by t3 for longer. Because the GA operates in action sets, classifiers which are not updated do not participate in reproduction and so cannot clone themselves. However, they may still gain numerosity as a consequence of other classifiers mutating or recombining to produce additional copies of them. It is unknown whether (or to what extent) this accounts for the difference in optimal delays for the two problems.

Preliminary work indicates that other dimensions also

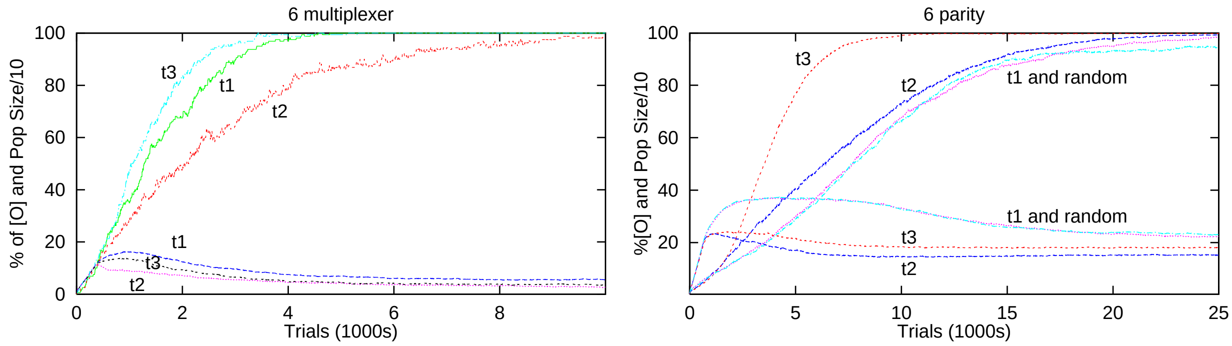

Figure 4: The upper three curves are the % of [O] while the lower three show population size. Multiplexer tests were averaged over 10 runs, parity over 30. t3's delay was 20 for multiplexer and 10 for parity tests.

affect the relative utility of deletion schemes and the optimal delay setting. One dimension is the ratio between problem space size and population size, which has a major effect on the importance of efficient deletion and hence the effect of t3. With an infinite population size our choice of deletion scheme is of no consequence. However, even increasing or decreasing the size limits used in this study by half would have a significant effect on the utility of t3 compared to the other schemes. We can think of t3 as making efficient use of classifiers when they are scarce.

Another factor which may be relevant is whether the problem is sequential or not. Sequential problems appear to differ greatly from single step problems even if they share the same number of states and useful generalisations (see [13]). Because of these considerable differences it seems likely that the optimal delay for t3 will also differ depending on whether the problem is sequential or not. However, this area remains unexplored.

We could recast the idea of protecting unevaluated classifiers as an exploration problem: how many trials should we allocate to a given classifier before we consider deleting it? This suggests that existing work in reinforcement learning on exploration (see [14] for a survey) is relevant to optimising deletion in classifier systems. In particular, work on distal exploration measures may be relevant in developing better deletion schemes for sequential problems.

3.2 A variation on t3

If t3 works because it provides some protection from the low-fitness bias to poorly evaluated classifiers, it may be beneficial to protect them from deletion completely. One variation on t3 is to assign any classifier which has been updated fewer than x times a deletion probability of 0, and to apply both biases from t3 to all others. However, this approach resulted in very bad performance on all measures on multiplexer tests with $x = 10$ and $x = 20$ and was abandoned. This approach may disrupt well-evaluated classifiers too much because it cannot delete poorly evaluated ones. Solutions would be to increase the population size limit, to add a mechanism which dynamically adjusts the size limit, or to introduce some mechanism by which the system could modulate the amount of protection given to new classifiers in order to balance the desire to protect them with the desire not to disrupt old ones.

A similar line of reasoning suggests that it may be beneficial to exclude classifiers from reproduction until they have been well evaluated in order to make sure that parents really are fit, but this too produced uniformly negative results on the multiplexer test. Genetic search appears robust enough that, for optimal search rates, a rapid turn-over in genetic material is preferable to accurate fitness evaluation.

4 Summary

The new deletion technique t3, which protects new classifiers from deletion, found the optimal solutions to the parity and multiplexer problems with the least computational effort of the techniques tested. Although this work has studied XCS, the delay principle of t3 should work in other evolutionary systems in which chromosomes cannot be fully evaluated immediately.

That XCS evolved complete optimal populations for the parity problem shows that it is able to learn functions with no or few useful generalisations despite its drive towards accurate generalisation. This is in contrast to Lanzi's finding that XCS's generalisation mechanism can make it unable to learn simple tasks in environments which afford little generalisation [15].

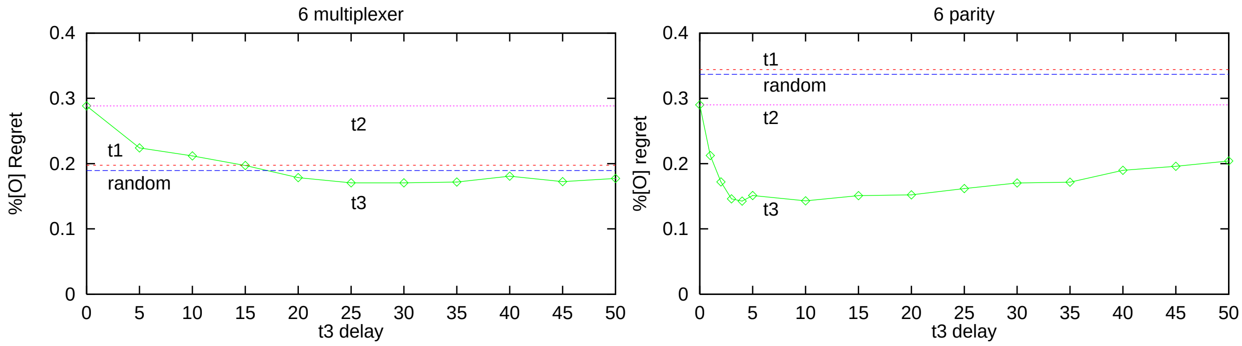

Figure 5: t3 achieves best %[O] regret across a range of delays on both problems, but makes more of an impact on the parity problem. Optimal delays are similar despite the difference in the problems. Multiplexer tests were averaged over 100 runs of 8,000 trials, parity over 30 runs of 25,000 trials.

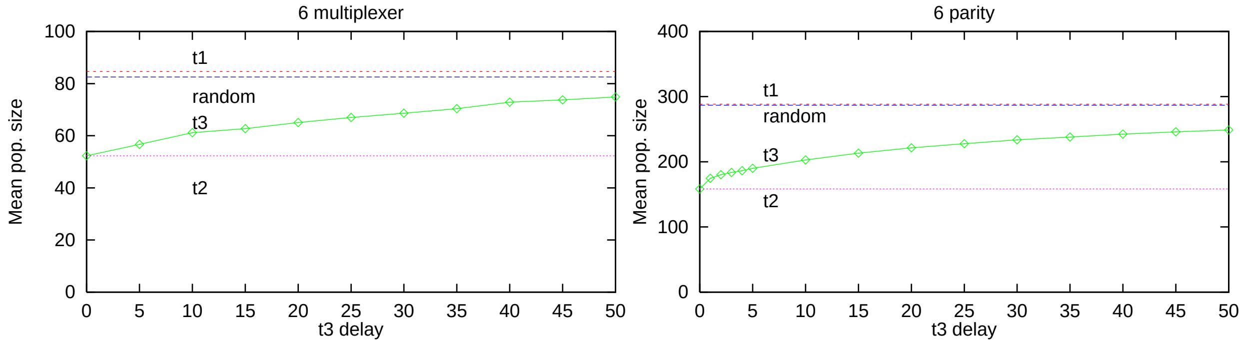

Figure 6: Population size increases steadily with t3's delay, but is still second best at delays which provide best performance.

A potentially significant difference between our studies is that Lanzi's Maze4 environment is a sequential problem while the parity function is single step. Further study of XCS's generalisation ability is required to resolve the issue.

Acknowledgements

I would like to thank Stewart Wilson, Manfred Kerber and Adrian Hartley for their comments and suggestions on this work and earlier drafts of this paper. This work was supported by the School of Computer Science at the University of Birmingham.

Appendix A - General System Settings

System settings for the multiplexer were: population size limit = 400, learning rate $\beta = 0.2$, crossover rate $\chi = 0.8$, mutation rate $\mu = 0.04$, hash probability $P_{\#} = 0.33$, low fitness deletion penalty threshold $\delta = 0.1$ (see 1.1), accuracy criterion $\varepsilon_o = 0.01$, accuracy falloff rate $\alpha = 0.1$, and GA threshold $\theta = 25$. The GA was conducted in the action sets and subsumption deletion was used except in figure 1. Population sizes were selected after experiments (not shown) suggested they were close to optimal for solving the problem as quickly and with as little computational effort as possible. "Trials" on the x axis of figures refers to the number of explore cycles using Wilson's pure explore/exploit scheme. Unless otherwise indicated t3's delay was 20 on multiplexer and 10 on parity tests. Other settings for the parity test were as for the multiplexer, except that the population size limit was 800. For explanation of these settings the reader is referred to one of [1, 11]. If we set $P_{\#}$ (the probability of creating a # at a given bit when creating a classifier condition) to 0, the parity problem becomes much easier to learn (with 2000 classifiers [O] is complete around 3,000 explore cycles rather than about 8,000). However, this sort of hand-tuning requires a priori knowledge of the nature of the problem, and in any case would not allow us to test XCS's robustness in terms of learning when useful generalisations are not available.

References

[1] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2), 1995. http://prediction-dynamics.com/

[2] Tim Kovacs. XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag, London, 1997. ftp://ftp.cs.bham.ac.uk/pub/authors/T.Kovacs/index.html

[3] Stewart W. Wilson. Classifier Systems and the Animat Problem. *Machine Learning*, 2:199–228, 1987.

[4] Stewart W. Wilson. Generalization in the XCS classifier system. In J. Koza et al., editor, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, San Francisco, CA, 1998. Morgan Kaufmann. http://prediction-dynamics.com/

[5] Thomas Bäck, David B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[6] Cees H. M. van Kemenade, J. N. Kok, and A. E. Eiben. Raising GA performance by simultaneous tuning of selective pressure and recombination disruptiveness. Technical Report CS-R9558, Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, August 31 1995. http://www.cwi.nl/static/publications/reports/CS-1995.html

[7] David E. Goldberg and Kalyanmoy Deb. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In Gregory J. E. Rawlins, editor, *Proceedings of the First Workshop on Foundations of Genetic Algorithms*, pages 69–93, San Mateo, July 15–18 1991. Morgan Kaufmann.

[8] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993. ftp://ralph.cs.cf.ac.uk/pub/papers/GAs/ga_overview1.ps

[9] Gilbert Syswerda. A Study of Reproduction in Generational and Steady-State Genetic Algorithms. In Gregory J. E. Rawlins, editor, *Proceedings of the First Workshop on Foundations of Genetic Algorithms*, pages 94–101, San Mateo, July 15–18 1991. Morgan Kaufmann.

[10] Lashon B. Booker. Triggered rule discovery in classifier systems. In *Proceedings Third International Conference on Genetic Algorithms (ICGA-3)*, pages 265–274. Morgan Kaufmann, 1989.

[11] Tim Kovacs. Evolving Optimal Populations with XCS Classifier Systems. Technical Report CSR-96-17 and CSRP-96-17, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996. ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSRP-96-17.ps.gz

[12] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993. ftp://ralph.cs.cf.ac.uk/pub/papers/GAs/ga_overview2.ps

[13] Tim Kovacs. Strength or accuracy? A comparison of two approaches to fitness calculation in classifier systems. In preparation, 1999.

[14] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4, 1996. HTML version: http://www.cs.brown.edu/people/lpk/rl-survey/rl-survey.html

[15] Pier Luca Lanzi. A Study of the Generalization Capabilities of XCS. In *Proceedings Seventh International Conference on Genetic Algorithms (ICGA-7)*. Morgan Kaufmann, 1997. http://www.elet.polimi.it/people/lanzi/index.html