evaluation involved testing the program on all 64 inputs from the 6-multiplexer domain, so that the total number of inputs was 15,680,000. This differs by a factor of 7,840 from the total number required by XCS. Thus, conservatively, the amounts of "experience" required by XCS and GP to learn the 6-multiplexer differ by three orders of magnitude.

There is not space to explore the reasons behind the difference. Instead, and in GP's favor, we note that the amount of computation per input is considerably larger for XCS, perhaps by enough to make the two approaches approximately equal in computational effort. For each input, GP has only to run the candidate program. In the case of the evolved 100% correct solution shown in Koza's Figure 7.41, only 3 "points" (`if` functions) were required. Many candidates were probably ten times larger, but it is clear that evaluation of the average candidate is quick.

In contrast, every input to XCS requires a "match step" in which the input is compared against the conditions of as many as several hundred classifiers. For the 6-multiplexer, at least, XCS clearly requires more processing per input than GP. But this will begin to offset XCS's enormous advantage in learning rate per input only if new inputs are available faster than XCS can process them. In some domains this will be the case; in others, such as robotic interactions with real environments, it will generally not be the case.

6 Conclusion

Subsumption deletion and an action set GA bring XCS close to the point of evolving populations consisting of accurate, maximally general, near-minimal covers of task environments, along with a residue of other classifiers of very low fitness. The ability to detect and represent environmental equivalences—to generalize accurately—is indispensable to creatures both natural and artificial. XCS appears to do this with tractable time and space complexity, and without needing, or being constrained by, pre-existing structure—classifiers simply evolve to suit the problem and its regularities. Many topics—for instance real-valued input vectors, incorporation of internal state, and application to robotics and other real-world domains—remain for future research.

Bibliography

Barto, A. G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology, 4*, 229-256.

Booker, L. B., (1982). *Intelligent behavior as an adaptation to the task environment*, Ph.D. Dissertation (Computer and Communication Sciences). The University of Michigan.

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

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

Cliff, D. and Ross, S. (1994). Adding temporary memory to ZCS. *Adaptive Behavior*, 3(2), 101-150.

Dorigo, M. and Bersini, H. (1994). A comparison of Q-learning and classifier systems. In D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson (eds.), *From Animals to Animats 3, Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, Cambridge, MA: The MIT Press/Bradford Books, 248-255.

Dorigo, M. and Colombetti, M. (1997). *Robot Shaping: An Experiment in Behavior Engineering*. Cambridge, MA: The MIT Press/Bradford Books, in press.

Fogarty, T. C. (1994). Co-evolving co-operative populations of rules in learning control systems. In Fogarty, T. C. (Ed.) *Evolutionary Computing, AISB Workshop Selected Papers, Lecture Notes in Computer Science 865*, Springer-Verlag, 195-209.

Holland, J. H. (1986). Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), *Machine learning, an artificial intelligence approach. Volume II.* Los Altos, California: Morgan Kaufmann.

Kovacs, T. (1996). *Evolving Optimal Populations with XCS Classifier Systems*. MSc. Dissertation, Univ. of Birmingham, UK.

Kovacs, T. (1997). XCS classifier system reliably evolves accurate, complete and minimal representations for Boolean functions. In *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing,* `http://www.cs.bham.ac.uk/~tyk/online97/online97.html.`

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

Lanzi, P. L. (1997). An analysis of the memory mechanism of XCSM. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference*. San Francisco, CA: Morgan Kaufmann.

Tufts, P. (in preparation). Doctoral dissertation, Computer Science Department, Brandeis University.

Wilson, S. W. (1994). ZCS: a zeroth level classifier system. *Evolutionary Computation, 2*(1), 1-18.

Wilson, S.W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation, 3*(2), 149-175.

the three tasks, i.e., jumping each time by a factor of about five. The final average values of M are 55, 148, and 345, respectively, but in the first two cases the values appear still to be falling. These numbers are good indicators of difficulty for the three tasks, but deriving from them a functional relationship is of course extremely uncertain. It seems quite clear, however, that the time and space complexity for XCS do not rise exponentially as does the size of the input space.

Each of the tasks admits a number of accurate maximal generalizations proportional to the number of terms in its DNF formula. For the 6-multiplexer, there are four terms. Each term leads to four generalizations, for a total of 16. For example, the term $x_0'x_1'x_2$ leads to the four maximally general classifiers:

000### : 0 $\Rightarrow$ 1000
000### : 1 $\Rightarrow$ 0
001### : 0 $\Rightarrow$ 0
001### : 1 $\Rightarrow$ 1000.

Similarly, the 11- and 20-multiplexers have eight and 16 DNF terms, and 32 and 64 generalizations, respectively. Thus the number of generalizations doubles in going from one task to the next more difficult one. The results of the previous paragraph suggest that difficulty as measured by arrival at 100% performance or final population size may also increase by a constant factor in going from one task to the next, though the factor is not equal to two. If true, this suggests that task difficulty is equal to the number of generalizations raised to a power, or $D = cg^p$, where D is a difficulty measure, g is the number of generalizations in the multiplexer, and c is a constant. Taking D equal to the number of problems for arrival at 100% performance, the data suggest $p = \log 5 = 2.32$ and $c = 3.22$.

The foregoing guess at a formula is highly speculative, since it is based on just three points and assumes that difficulty increases by a constant factor between tasks. However, the guess suggests that the "learning complexity" of the multiplexer functions for XCS is of order equal to the number of generalizations raised to a constant power, or polynomial in g. A low-power (~2) polynomial dependence on generalizations implies good prospects for scale-up to realistically large environments. It is important to see if this relationship holds for larger multiplexers (the next has 37-bit strings).

The complexity can also be directly expressed in terms of l, the string length. To do this, note that as multiplexers get bigger, $l \cong 2^k$. The number of generalizations g is proportional to 2^k, so, as k increases, l becomes proportional to g. Thus, under the assumptions made earlier, since D is polynomial in g, it is also polynomial—not exponential—in l.

The multiplexer—and woods—experiments indicate that XCS has a strong tendency to detect the regularities of an environment and to represent them efficiently. These characteristics are essential for systems that must learn in realistic environments. We predict that XCS—with suitably modified classifier syntax (Wilson 1995)—will show similar representational efficiency in environments with continuous sensory inputs.

5.2.3 Comparison with genetic programming

Koza (1992, pp. 191-198) uses genetic programming (GP) to evolve programs that correctly compute the 6-multiplexer function. GP exemplifies *phylogenetic adaptation*. A population of candidate "creatures"—in this case Lisp programs—is evolved over generations, analogously to natural evolution of species. Each program is supposed to compute the correct multiplexer value for any input from the problem environment.

In contrast, a classifier system such as XCS exemplifies *ontogenetic adaptation*. The classifier system is analogous to a *single* creature adapting over its lifetime. There is an evolving population inside the classifier system, but its members (the classifiers) consist of partial, not whole, solutions to the problem posed by the environment. The object of the system is to evolve classifiers such that, among them, all environmental inputs are dealt with optimally according to some criterion. In the case of XCS and the 6-multiplexer the criterion is that the system predict correctly the payoff corresponding to each possible input and each possible decision by the system. In the case of payoff 1000 for the correct function value and payoff 0 for incorrect, correct prediction is equivalent to producing the correct value of the Boolean function.

Thus GP and XCS solve the 6-multiplexer, but in very different manners. GP evolves individuals that are total solutions to the problem. XCS evolves partial solutions which, taken together, solve the problem. It is of considerable interest to compare the efficiencies of the two approaches.

A full comparison is beyond the present scope, since it would involve, among other things, comparison over a range of problem types and detailed implementation information. However, for the 6-multiplexer it is possible to compare results on the number of problem inputs (6-bit strings) required for each system to "learn" the function, and to comment on the amounts of computation effort.

Figure 6, which averages 10 runs, shows that XCS reaches nearly 100% performance having seen approximately 2,000 randomly selected inputs. Summarizing run statistics on the 6-multiplexer, Koza's (1992) Figure 8.4 indicates that under appropriate parameters for population size, generations per run, and number of runs, GP was able with 99% probability to evolve at least one 100% correct solution after evaluating as few as 245,000 individual candidate programs. Each such

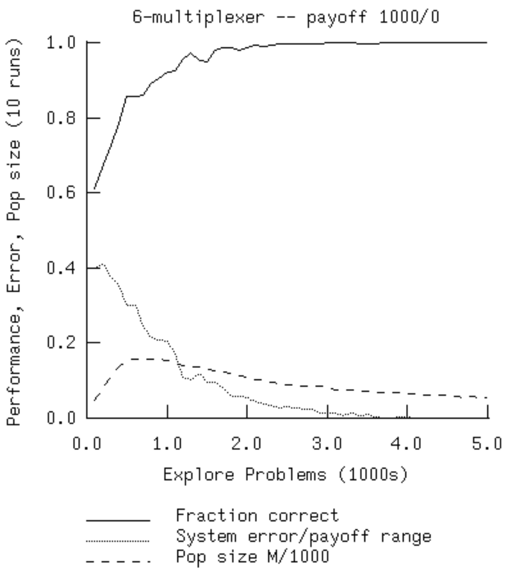

Figure 6. 6-multiplexer with 1000/0 payoff.

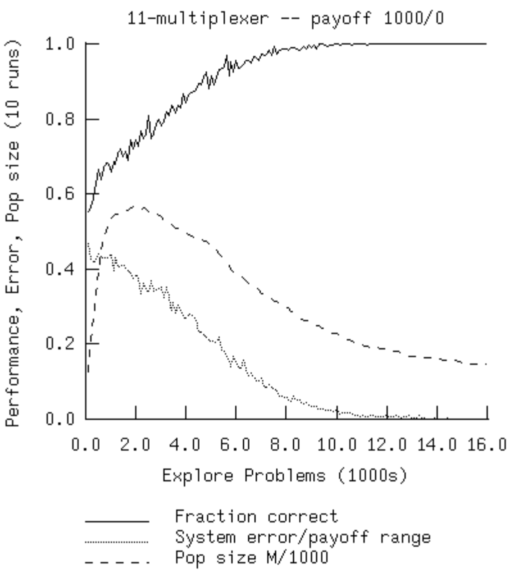

Figure 7. 11-multiplexer with 1000/0 payoff.

spond to the terms in the function's formula, thus permitting the formation of generalizations if XCS can detect them. As with the Woods2 experiments, test problems alternated with pure explore problems.

As seen in Figure 5, moving the GA to [A] from [M] does not significantly affect the population size, but the effect of the subsumption deletion technique is to halve the size. This further supports the motivation for doing the GA in [A]. It was noted in Section 4.1 that the argument for moving to [A] does not apply if X x A for the task has certain symmetries. In fact, the multiplexers have the symmetries, namely that if, e.g., the classifier 000###:0 is an accurate maximal generalization, then the classifier 000###:1 is also an accurate maximal generalization, albeit predicting a different payoff than the first. Thus the Woods2 and multiplexer results are consistent with the theory presented in Section 4.1, and indicate that, in general, it is better to do the GA in the action sets.

5.2.2 *Complexity indications*

Incorporating the two improvements in XCS, we re-did the multiplexers for the classic task in which payoff is 1000 for the correct answer and 0 for the wrong answer; that is, a finite payoff for correct and nothing for incorrect. This is the payoff schedule usually associated with the problem. Our objective was to produce a benchmark for future comparisons, and to estimate the rate of growth of difficulty for XCS as problem size increased. The experiments were carried out on the 6-, 11-, and 20-multiplexers, with results as shown in Figures 6, 7, and 8. Parameter settings were similar to those used in Wilson (1995).

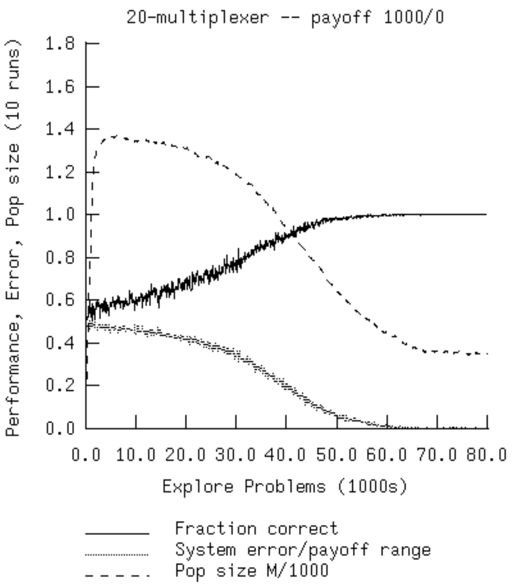

Figure 8. 20-multiplexer with 1000/0 payoff.

Please observe the scale differences for the three figures. They were plotted so as to situate the arrival at 100% performance near the middle of the graph. Fraction correct is simply the fraction of the past 50 test problems for which XCS gave the correct answer. System error/payoff range is the absolute error in the system's payoff prediction divided by 1000. Pop size is *M*, the size of the population in macroclassifiers, starting with an empty population.

Very roughly, XCS reaches 100% performance at about 2,000, 10,000, and 50,000 problems respectively in

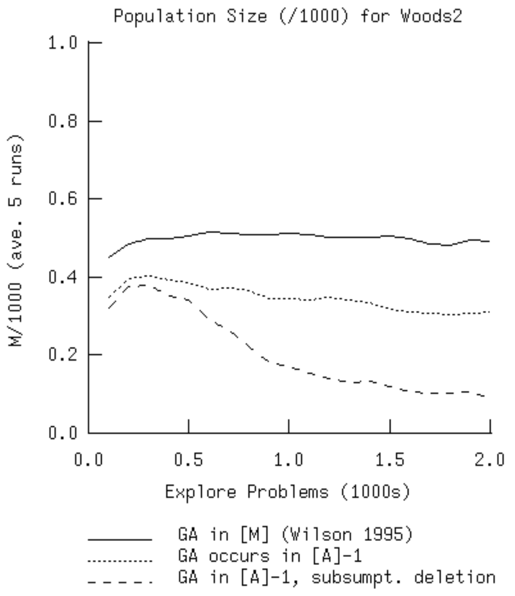

Figure 4. Population size comparison on Woods2.

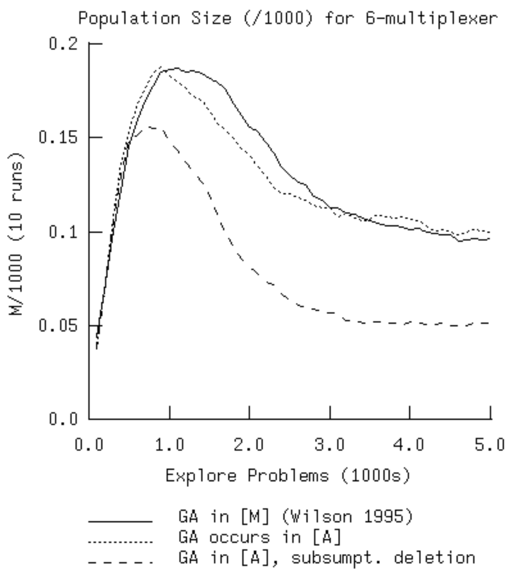

Figure 5. Population size comparison on 6-multiplexer.

until it appeared that M would not decrease further, then examining the population. M fell to between 50 and 60, after which it oscillated slowly. Examination of the populations revealed that approximately 35 classifiers had high fitness and numerosity, while the rest had sharply lower fitness and numerosity, and were very "young"—thus recently generated by the GA and "in the pipeline" to deletion. Since performance was still at optimum, a minimal cover for Woods2 cannot require more than about 35 classifiers. The regions of X x A covered by the high fitness classifiers were examined in detail, with the observation that there was relatively little overlap between the regions. From this we concluded tentatively that a minimal cover for Woods2 requires about 35 classifiers. Interpretation of the conditions of the high-fitness classifiers was generally straightforward: concepts like "anywhere along the left side of a block (of objects)", "blank below", etc., are easily identified. Our tentative conclusion is that the two improvements permitted XCS to converge on an essentially minimal cover, plus a small number of additional classifiers whose low fitness identifies them as ignorable.

5.2 Multiplexers

Boolean multiplexer functions are defined for binary strings of length $l=k+2^k$. The function's value may be determined by treating the first k bits as an address that indexes into the remaining 2^k bits, and returning the indexed bit. For example, in the 6-multiplexer ($l=6$), the value for the input string 100010 is 1, because the "address", 10, indexes bit 2 of the remaining four bits. In disjunctive normal form, the 6-multiplexer is fairly complicated (the primes denote negation):

$$F_6 = x_0'x_1'x_2 + x_0'x_1 x_3 + x_0x_1'x_4 + x_0x_1x_5.$$

F_{11} and F_{20} are the next more complicated multiplexers. The corresponding expression for F_{11} has eight terms each consisting of four factors; for F_{20} there are 16 terms of five factors each. The multiplexers have been employed as test problems in many learning investigations beginning with Barto (1985). Though they are "one-step"—i.e., not sequential—artificial environments, we employ them because they permit study of generalization without the added complications that sequential environments entail. The multiplexers are highly nonlinear—thus difficult—and the "family" of multiplexers permits testing one's system against a series of increasingly difficult yet formally related tasks. In fact, because as l increases the task stays similar but the input space grows exponentially, the multiplexers offer a way of evaluating the necessary complexity of a learning system as a function of l. A learning system that simply memorized the answer for each input would of course grow exponentially, too. On the other hand, a learning system that like XCS is capable of generalization should grow more slowly with l, and it is of interest to estimate the rate.

5.2.1 6-Multiplexer population comparisons

We begin by comparing results on the 6-multiplexer using the same three regimes as in Section 5.1. Figure 5 compares the population sizes as a function of explore problems for the three regimes. The particular task in this case is the same as in Wilson (1995, Figure 3). In each problem, a random string is presented to XCS and it chooses an "answer" (1 or 0). A payoff is then given the system that depends on the subspace of X x A to which the input belongs, but is larger for the correct answer than the wrong answer. The subspaces corre-

```
..............................
.QQF..QQF..OQF..QQG..OQG..OQF.
.OOO..QOO..OQO..OOQ..QQO..QQQ.
.OOQ..OQQ..OQQ..QQO..OOO..QQO.
..............................
..............................
.QOF..QOG..QOF..OOF..OOG..QOG.
.QQO..QOO..OOO*.OQO..QQO..QOO.
.QQQ..OOO..OQO..QOQ..QOQ..OQO.
..............................
..............................
.QOG..QOF..OOG..OQF..OOG..OOF.
.OOQ..OQQ..QQO..OQQ..QQO..OQQ.
.QQO..OOO..OQO..OOQ..OQQ..QQQ.
..............................
```

Figure 2. Environment "Woods2" with animat. Empty cells are indicated by "."

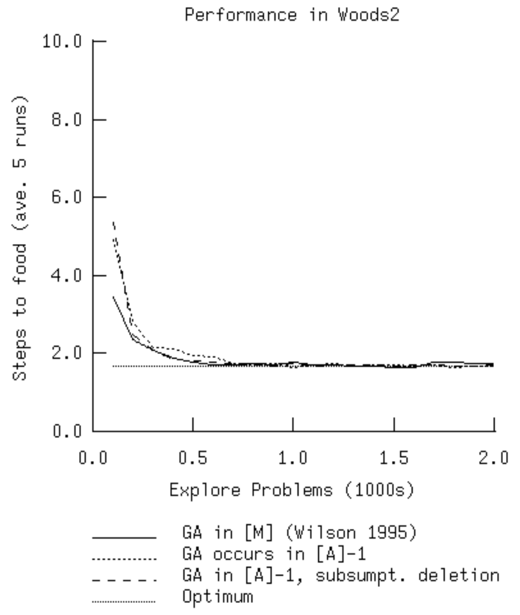

Figure 3. Performance comparison on Woods2.

to real environments. The latter may have some special issues involving Q-like learning in continuous spaces (Cliff & Ross 1994) that will have to be confronted *in situ*, so to speak. However, these issues do not include adapting the classifier format for continuous variables, which seems fairly clear and is sketched in Wilson (1995)).

The action-selection regime for the experiments had two modes, "pure exploit" and "pure explore". In pure exploit, XCS chooses on each time-step the action for which the predicted payoff is highest. In pure explore, XCS chooses an action randomly. Each run of an experiment had 2,000 "explore problems", in each of which the animat was started in a random open position then allowed to move under pure explore until food was encountered. After each explore problem, the animat was again randomly started, but carried out a test problem in which it ran in pure exploit until food was found. Test and pure explore problems alternated. The system's performance was a moving average of the previous 50 test problems.

Figure 3 shows performance curves from experiments carried out under three regimes. The first listed is that of Wilson (1995). The second is the same, except that the GA is conducted in the action sets instead of the match sets. The third is like the second, with the addition of the "subsumption deletion" technique of Section 4.2. Note that the three curves are quite similar, except for a slightly faster initial improvement under the first regime. Also shown is the optimum performance for Woods2: an average of 1.7 steps to food from a random start. In contrast, the average number of steps if moves are random is about 27 steps; all three curves start around this level, falling rapidly in the first 100 problems to the levels plotted. The three regimes thus have about equal performance in Woods2, reaching close to the optimum by about 500 problems.

In contrast, population size results for the three regimes are quite different (Figure 4). In the three cases the population was initially empty (the first few classifiers were created by a cover operation). In the original experiment, *M*, as shown, increased rapidly to around 500, staying near there for the rest of the experiment. Woods2 contains exactly 70 distinct input vectors, so with eight possible actions, tabular Q-learning would require a table of size 560 for Woods2. Classifier systems, XCS in particular, are capable of generalization, so one would hope to learn the woods with less than one classifier for each state-action pair. In fact, 500 is less than 560, but not by much.

Changing the GA to the action sets reduced the maximum level of *M*, and brought the value to 310 by 2,000 problems, a respectable improvement that supports the motivation of Section 4.1 for moving the GA from the match sets to that action sets . However, the addition of subsumption deletion brought *M* down farther, to 89 at 2,000 problems. It is of interest to know how *few* classifiers are sufficient in Woods2 to cover X x A while preserving optimal performance. Woods2 is quite complicated, and just which classifiers would constitute a minimal cover is not obvious. Presumably there are several possible minimal or near-minimal covers. An estimate of the size of a minimal cover was obtained by continuing the third experiment beyond 2,000 problems

ample (please see Wilson (1995) for the full context), the classifier ######################## : 1
is accurate and maximally general in Woods2, but if the same condition is attached to any of the other seven actions, the result is no longer accurate. (In this and subsequent classifier examples, we will sometimes omit the prediction part ("$\Rightarrow$ <*prediction*>") for simplicity.)

The implication was that if a given match set is to contain accurate, maximally general classifiers for each of the possible actions, they would often have *different* conditions. But if that is the case, crossover would not be benign: it might easily generate inaccurate classifiers by crossing parents with different actions. For this reason we began to experiment with doing the GA in the action set instead of the match set.

4.2 Subsumption deletion

The other problem noted above was the presence of accurate, but unnecessarily specialized classifiers. Here is a toy example. Suppose the classifier C2 = 1###:3, is accurate and maximally general in some environment. (Recall that "accurate" means having error less than ε_0; "maximally general" means you can't change any 1 or 0 to # without making the classifier inaccurate.) The classifier C1 = 11##:3, is also accurate, since it is subsumed by C2, but it is not maximally general, since C2 is. The problem was that classifiers like C1 were often present in evolved populations, whereas they are completely unnecessary.

Investigation revealed a subtle distinction. Sometimes C1, being more specialized, would match in fewer situations than C2, and so would eventually be eliminated by the GA. Its presence simply reflected the "waiting time" till deletion. In other cases, however, it was *not* the case that C1 matched in fewer cases than C2. The environment was such that every situation matched by C2 was also matched by C1. This could occur if the environment was "sparse", i.e., the number of input strings that actually occurred in the environment was less than the number permitted by the coding (as is nearly always the case in real environments). Thus C2 was *formally* more general than C1, but this fact was not reflected in the actual frequencies of matching inputs. As a result, C2 and C1 had equal fitnesses and sat on the same fitness plateau, so to speak.

Wilson (1995) described the second "1" in C1's condition as an "optional" bit, since changing it to # did not affect fitness. The first bit, on the other hand, is *not* optional—changing it to # results in inaccuracy; it is "essential". We sought a technique that would in sparse environments eliminate optional bits while preserving essential ones.

A first approach was to change the fitness function so that if two classifiers were *accurate*, but one was formally more general than the other, the former would have a higher fitness. This technique was in the right direction, but a more powerful one was found, as follows. When an offspring is generated by the GA, its parents are examined to see if either of them is both accurate and its condition logically subsumes the condition of the offspring. The meaning of subsumption here is that the set of possible strings matched by the offspring is a proper subset of the set matched by the parent. If this test is satisfied, the offspring is abandoned (not injected into the population) and the numerosity of the parent is incremented by one. A similar check for subsumption was done in action sets since an occasional offspring would be generated that was subsumed by some classifier *elsewhere* in the population. In addition, the test was extended to require that any subsuming classifier have an experience value greater than a threshold (e.g., 20), to insure that its accuracy had been sufficiently estimated.

The foregoing method of "subsumption deletion" may be viewed genetically as a kind of directed mutation. In effect, for parents estimated to be accurate, the GA is constrained to generate and evaluate only offspring that are more general than the parents.

5 Results of Experiments

The two prospective improvements were tested on the "Woods2" and multiplexer tasks of Wilson (1995). In addition, earlier software problems were overcome permitting the first XCS experiment on the 20-multiplexer. Because the present objective is to compare with earlier results, the descriptions of the tasks and input codings will be abbreviated; please see Wilson (1995) for details.

5.1 Woods2

Woods2 (Figure 2) is a grid-like environment in which an "animat" (shown by *) starts at a random open position and moves under control of XCS until a food object (F or G) is bumped into, at which point it receives a reward of 1000 and is restarted randomly. A move into a "rock" (O or Q) is not allowed to take place, but the step is counted nevertheless. The animat's sensory input consists of a 24-bit vector, with three bits coding features of the object (including the blank object) observed in each of the eight cells surrounding the animat. The animat's available actions consist of moves into the eight cells.

(Use of grid-like environments and discrete inputs and actions is obviously a simplification from real environments in which artificial creatures should eventually operate. Our view is that the core problems of learning involve prediction, generalization, and internal state, and that these can be much more easily studied in discrete environments with results that will later translate single structure, instead of being dispersed. As a population converges on increasingly general, accurate classifiers, its size in macroclassifiers, M, decreases. Thus M is a measure of the space complexity of the classifier system. It will be referred to further in the rest of the paper. XCS also has a parameter N which determines the (constant) effective length of the population in microclassifiers (ordinary classifiers), and corresponds to the traditional fixed size of a classifier population. XCS's deletion regime (not detailed here) ensures that the sum of the numerosities of the population's macroclassiers does not exceed N.

Other points of difference between XCS and traditional classifier systems include: XCS's Q-learning-like update regime (which is different from, but closely related to the traditional bucket-brigade algorithm (Dorigo & Bersini 1994, Wilson 1994)); an action-selection step that is not committed to a specific mechanism as is, e.g., the traditional "roulette wheel" action selection; and a deletion mechanism that keeps resources balanced among niches (Booker 1989).

3.2 Generalization mechanism

In XCS, classifiers that predict more accurately get higher fitness. If a classifier's condition is highly specific, one would expect it to be more accurate than one with a less specific condition, since it has to predict the payoff over a smaller number of situations. If fitness is based on accuracy, more-specific classifiers should win out over less-specific ones. Why then should XCS drive, as advertised, toward *general* accurate classifiers?

The answer lies in the observation that, under a genetic algorithm, reproductive success depends not only on fitness but also on reproductive opportunity. Consider two (micro)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 C1 and C2 have the same ε, and are thus equally accurate. Which will win out?

Every time C1 and C2 occur in the same action set, their fitness values will be updated by the same amount. However, because C2 is a generalization of C1 it will tend to occur in *more* match sets than C1, and thus probably (depending on the action-selection regime) in more action sets. Because the GA occurs in action sets, C2 will have more reproductive opportunities and thus its number of exemplars will tend to grow with respect to C1's (in macroclassifier terms, 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 (the κ_j' sum to 1.0) would be "steered" toward exemplars of C2, resulting via the GA in yet more exemplars of C2 relative to C1. Eventually, C2 will displace C1 from the population.

Thus XCS does drive toward accurate classifiers, but any accurate classifier will be beaten out be a more general version of itself that maintains the same accuracy. The process will continue until further generalization results in a loss of accuracy. This point depends on the value of ε_0 in the accuracy function. Since accuracy falls exponentially if error is greater than ε_0, classifiers will evolve to be as general as possible while still having errors less than ε_0. If they become more general still, their chances of survival are greatly diminished. Kovacs (1997) suggests that XCS's generalization mechanism drives toward populations that are optimal representations.

4 Improvements to XCS

As reported in Wilson (1995), experiments with XCS on sequential and one-step reinforcement learning tasks demonstrated both the system's performance success and its drive toward accurate general classifiers. However, in terms of generalization, the system as it then stood had shortcomings. Examination of evolved populations showed that the most general accurate classifiers were accompanied by numerous others. Many were specializations of the former—why? There were also many inaccurate classifiers. It was clear that the populations were not as "condensed"—i.e., M was not as small—as might be desired. For instance, what prevented a population from condensing until it consisted solely of accurate, maximally general classifiers sufficient to cover X x A?

4.1 From GA in [M] to GA in [A]

A first step was to address the existence of inaccurate classifiers. A clue was that they were usually "young", as indicated by an experience parameter that counts the number of times a classifier occurs in an action set. This suggested that most had recently been generated, and that the GA would eventually eliminate them. But why were they being generated in the first place? The GA then occurred in the match sets, and all match set classifiers have to match the same input, so their conditions are similar. In fact, if the GA is really working as predicted, shouldn't each match set evolve to have one macroclassifier per action, with identical maximally general conditions? Any other classifiers in the match set would be due to occasional mutation noise, since crossover could not create anything new from classifiers with identical conditions.

Study suggested that it is incorrect to assume that if a classifier with action a has a maximally general, accurate condition, then a classifier with the same condition but a different action must also be maximally general and accurate. Such a rule holds if a task has the right symmetries, but is not true in general. In the Woods2 task we found many cases where it was not true. For ex-

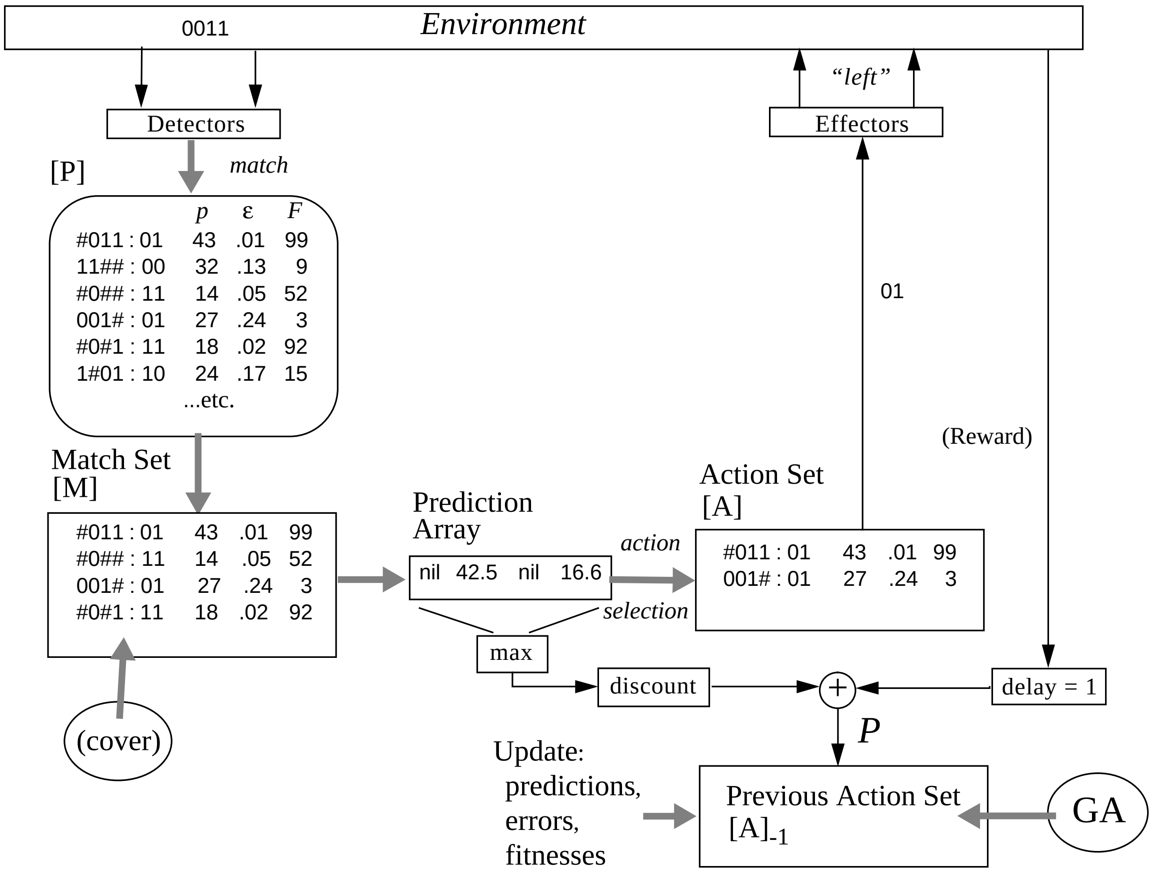

Figure 1. Schematic illustration of XCS (showing new GA site).

tems. The idea of performing the GA in a niche (i.e., a set of environmental states each of which is matched by the same set of classifiers) instead of the whole population was introduced by Booker (1982) (see also Fogarty (1994)). XCS first followed Booker in performing the GA in the match set [M], but the GA now takes place in the action sets $[A]_{-1}$ or [A]. As will be explained in Section 4, this change increases the proportion of accurate, maximally general classifiers in the population.

The basic idea of a niche instead of panmictic GA eliminates the undesirable competition that otherwise occurs between classifiers in different match sets of a common, cooperative chain of actions. In addition, as Booker pointed out, crossovers within a niche are more likely to yield useful classifiers than crossovers between potentially unrelated classifiers that match in different niches. However, even with a niche GA, there remains the problem that when fitness is based on strength (or, in effect, payoff prediction) as in traditional classifier systems, the GA cannot distinguish an accurate classifier with moderate payoff from an overly general (i.e., inaccurate) classifier having the same payoff on the average. The result is reduced performance due to errors by the latter (Cliff & Ross, 1994).

Macroclassifiers. Whenever XCS generates a new classifier, the population is scanned to see if the new classifier has the same condition and action as any existing classifier. If so, the new classifier is not actually added to the population, but a *numerosity* field in the existing classifier is incremented by one. If, instead, there is no existing classifier with identical condition and action, the new classifier is added to the population with its own numerosity field initialized to one. We term such classifiers *macroclassifiers*. They are essentially a programming technique that speeds up matching and other aspects of processing. They do not affect the underlying "microclassifier" operation of the system, because all system functions are written so as to take into account the numerosities (Kovacs (1996) evaluates macro- vs. microclassifiers). Traditional classifier systems could also implement macroclassifiers.

However, experience with macroclassifiers has been very helpful in understanding the evolution of populations, since equivalent classifiers are represented by a

A corollary is that generalization will not be possible if the creature's internal representational architecture or language cannot express it—e.g., if the temperature creature can't use rules of the kind above—even though equivalent situations exist in the environment. The ability to generalize depends both on a system's ability to detect equivalences in the environment, and to express them using its representational equipment.

Generalization is important, indeed vital, for learning systems because they depend on vectors of sensor readings from an input space X whose size is exponential in the number of sensors. Fortunately, most real environments have regularities. The input space, or more precisely the product space of X and the available action set A, X x A, often contains regions of equivalent consequence for the creature, permitting generalization if it can detect and represent the regions. In fact, for systems of interest, X x A may be extremely large, so generalization is essential for practical artificial systems. For the same reason, an essential part of understanding animals will be to understand their processes of generalization.

In a classifier system, generalization occurs whenever a classifier is capable of matching more than one input vector. For example, the (XCS) classifier 10##:1 $\Rightarrow$ 700, with "#" a "don't care" symbol, predicts that taking action "1" in response to any of the (four possible) input vectors beginning with "10" will result in a payoff of 700. This classifier makes what we term a *generalization* with respect to the subspace 10## x 1 of X x A. The asserted generalization may or may not be *accurate*, depending on the degree to which the payoff actually received varies from 700 within the subspace.

Via the standard {1,0,#} syntax of conditions, classifiers can express conjunctive generalizations, which correspond to terms in the disjunctive normal form (DNF) of a Boolean formula. Via more general syntax—e.g., Lisp s-expressions and operators drawn from genetic programming (Wilson 1994, Tufts in preparation)—they may express arbitrary generalizations. Interestingly, since the system's generalizations, or concepts, are expressed by individual classifiers, it can be relatively easy to "see the knowledge"—more so than in learning systems such as neural networks in which knowledge is represented more diffusely. To the extent the classifier system evolves accurate, maximally general classifiers, the concepts it has learned will be even clearer. Of course, *our* being able to see the knowledge is irrelevant from the viewpoint of the creature itself. But further development of artificial learning systems may involve introspective operations, in which it will be useful for the system itself to examine what it knows. Then the perspicuity of classifiers, especially if they are accurate and general, could be vital.

3 XCS & its generalization mechanism

3.1 Brief review of XCS

XCS has many resemblances to more usual classifier systems (Holland 1986), but differs in key respects. A detailed discussion is given in Wilson (1995). XCS employs standard classifier condition-action rules (internal state such as a message list has not yet been implemented, but see Lanzi (1997)), updates parameters of the classifiers each time they are active, and uses a genetic algorithm (plus rare input covering) to generate new classifiers. The principal differences are as follows (see Figure 1).

New classifier parameters. Each classifier has three associated parameters: *prediction* p_j, *error* ε_j, and *fitness* F_j. The "strength" parameter is gone. The prediction p_j is a statistic estimating the Q-learning-like payoff P (see Figure 1 for the computation of P) when that classifier matches and its action is chosen by the system; it is updated according to $p_j \leftarrow p_j + \beta (P - p_j)$, where β ($0<\beta\leq1$) is a learning rate constant. The error ε_j is an estimate of the error in p_j; it is updated according to $\varepsilon_j \leftarrow \varepsilon_j + \beta (|P - p_j| - \varepsilon_j)$. The calculation of fitness F_j is more complicated.

As noted in the Introduction, a classifier's fitness is based on the accuracy of its payoff prediction. The fitness is arrived at in steps. First, the classifier's *accuracy* κ_j is computed using the formula

$\kappa_j = \exp[(\ln \alpha)(\varepsilon_j - \varepsilon_0)/\varepsilon_0)]$ for $\varepsilon_j > \varepsilon_0$, otherwise 1.

In this expression, ε_0 is a threshold such that if the classifier's error is less than ε_0, the classifier is deemed "accurate", and given accuracy 1. If, on the other hand, the error is greater than ε_0, the resulting lower accuracy is given by a falling exponential of the error ($0 < \alpha < 1$). A negative power-law function of the error may also be employed.

In the second step are calculated the κ_j of all the classifiers in the action set (the previous action set $[A]_{-1}$ for sequential problems, the current action set [A] for one-step problems; see Figure 1). Then a *relative accuracy* κ_j' is calculated for each classifier in the action set by dividing its κ_j by the sum of the κ_js of the set. Finally, the fitnesses of each classifier in the set are updated according to $F_j \leftarrow F_j + \beta (\kappa_j' - F_j)$.

Every time a classifier takes part in an action set, p_j, ε_j, and F_j are updated. The net effect is for the fitness of a classifier to represent the accuracy of its payoff prediction relative to the prediction accuracies of other classifiers that typically occur in its action sets. This is the basis for the selective pressure in XCS toward more accurate classifiers.

A niche GA. In the original XCS, the genetic algorithm was executed in the match set instead of panmictically using the whole population as in many classifier sys-

From *Genetic Programming 1998: Proceedings of the Third Annual Conference*, J. Koza et al. (Eds.)
San Francisco, CA: Morgan Kaufmann

Generalization in the XCS Classifier System

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

Abstract

This paper studies two changes to XCS, a classifier system in which fitness is based on prediction accuracy and the genetic algorithm takes place in environmental niches. The changes were aimed at increasing XCS's tendency to evolve accurate, maximally general classifiers and were tested on previously employed "woods" and multiplexer tasks. Together the changes bring XCS close to evolving populations whose high-fitness classifiers form a near-minimal, accurate, maximally general cover of the input and action product space. In addition, results on the multiplexer, a difficult categorization task, suggest that XCS's learning complexity is polynomial in the input length and thus may avoid the "curse of dimensionality", a notorious barrier to scale-up. A comparison between XCS and genetic programming in solving the 6-multiplexer suggests that XCS's learning rate is about three orders of magnitude faster in terms of the number of input instances processed.

1 Introduction

XCS, a recently developed classifier system (Wilson 1995), focused on the question of how the central quantity in the system's evolutionary process, classifier fitness, should be defined. The answer, a fundamental change, is important not only because it resulted in better performance, but because it gave the classifier system, for the first time, a substantial and theoretically grounded generalization ability.

XCS's definition of fitness is different from that of traditional classifier systems. The new fitness is based on the *accuracy* of a classifier's payoff prediction, instead of the prediction itself, or strength. Combined with use of a niche, instead of a panmictic, genetic algorithm (GA), the new fitness results in a strong tendency to evolve classifiers that are not only accurate but maximally general. Previous classifier systems did not tend to produce accurate general classifiers (Cliff & Ross, 1994).

The present paper reports further improvements in XCS's generalization ability. One improvement is to conduct the genetic algorithm in the action sets instead of the match sets. The other is a technique in which an offspring classifier whose condition is subsumed by that of an existing accurate classifier is replaced by a clone of the subsuming classifier. The improvements increase the proportion of the population that consists of maximally general, accurate classifiers, at the same time reducing the total number of classifiers—at no reduction in performance. A classifier is a basic kind of concept—it expresses the payoff to be received when its action is taken in situations covered by its condition. The techniques now in XCS advance classifier systems' ability to generate and directly express accurate, maximally general concepts about their environments.

The next section briefly discusses generalization and its desirability in natural and artificial learning systems. Section 3 reviews the underlying generalization mechanism in XCS. Section 4 describes the two improvements in detail. Results of experiments with the multiplexer and "woods" problems of Wilson (1995) are presented in Section 5, together with a discussion of learning complexity and a comparison with multiplexer learning by genetic programming. Section 6 concludes.

2 Generalization

Briefly, generalization means to treat as equivalent, differently appearing situations that nevertheless have equivalent consequences for the learning system. Implied—and necessary—in this is the idea that the system not only knows the equivalence, but deals with it "compactly". For example, imagine a creature that is active at temperatures below T_{crit} and sleeps above. If it were using a fine-grained lookup table to choose its behavior, with a slot for each half-degree celsius, we would not regard the creature as generalizing much. We *would* regard it as generalizing if, e.g., the system used a *rule* such as (**if** $T < T_{\mathrm{crit}}$ **then** active **else** sleep), that represents the conditions for the behaviors with some degree of compactness. Thus, generalization means to recognize environmental situations having equivalent consequences, but to do so using internal structure of significantly less complexity than the raw environmental data.