A Hierarchical XCS for Long Path Environments

Dr Alwyn Barry

University of the West of England,
Coldharbour Lane, Bristol, BS16 1QY, UK

Email: alwyn.barry@uwe.ac.uk
Phone: (++44) 344 3135

Abstract

It has been noted (Lanzi, 1997, Butz et al, 2000) that XCS (Wilson, 1998) is unable to identify an adequate solution to the Maze14 problem (Cliff and Ross, 1994) without the introduction of alternative exploration strategies. The simple expedient of allowing exploration to start at any position in the Maze will allow XCS to learn in such 'difficult' environments (Barry, 2000b), and Lanzi (1997) has demonstrated that his 'teletransportation' mechanism achieves similar results. However, these approaches are in truth a re-formulation of the problem. In many 'real' robotic learning tasks there are no opportunities available to 'leapfrog' to a new state. This paper describes an initial investigation of the use of a pre-specified hierarchical XCS architecture. It is shown that the use of internal rewards allows XCS to learn optimal local routes to each internal reward, and that a higher-level XCS can select over internal sub-goal states to find the optimum route across sub-goals to a global reward. It is hypothesised that the method can be expanded to operate within larger environments, and that an emergent approach using similar techniques is also possible.

1 INTRODUCTION

Within their investigation of the introduction of a memory mechanism to ZCS, Cliff and Ross (1994) introduced the Maze 14 environment. This Markovian environment provides a length 18 corridor path constructed using the Woods-1 inputs, providing a non-linear action route to the reward position. The length of the pathway which the environment provides is itself a challenge to current Bucket-Brigade algorithms. For example, Riolo (1987) estimated that for CFSC within single action corridor environments the number of times payoff must pass through the rule-chain to achieve 90% of stable strength is $R = 22 + 11.9n$[1]. In such an environment using pure random exploration the probability of exploring state s_n from state s_{n-1} is always P(1.0). However, within the Maze14 environment the probability is P(0.125). Clearly this probability remains the same within each successive state, and thus, the probability of moving directly from s_0 to s_{18} in 18 steps is P(5.55×10^{-17}). It is therefore highly improbable that ZCS will move from s_0 to s_{18} within exploration even where a large number of iterations [in bucket-brigade terms] is permitted in each trial. Furthermore, transitions from s_n to s_n will be explored with P(0.875), resulting in a disproportionate exploration of the early states within this environment.

The problem of exploration within the Maze14 environment can be overcome by a number of mechanisms. The first involves a change in the environment definition itself to allow all of states s_0 to s_{17} to be start states. This means that exploration starting in the later states will have a higher chance of reaching the reward state to feed back stable reward. This reward will be passed down the local states of the chain which when subsequently explored will further propagate these values. XCS (Wilson, 1998) allows prediction learning within exploitation so that the pathway to the reward state will rapidly become established through exploitation once discovered. Lanzi's 'teletransportation' mechanism (Lanzi, 1997) is an example of the use of this approach.

An alternative approach that does not require a change in the environment definition would be to dynamically modify the division between exploration and exploitation so that as transitions within the FSW are increasingly explored their probability of future exploration is decreased. This allows the LCS to advance progressively further through the environment to areas that require exploration. Lanzi (1997) notes that Wilson has used this approach within Maze-14, although no results have been published to date.

It is possible that a third approach to this problem exists if the LCS representation can be expanded to allow a hierarchy of classifiers to be utilised. This paper elaborates a hypothesis for the use of a hierarchy for the solution of such problems and investigates the use of the approach identified within a number of corridor environments. It provides a number of new contributions to LCS research. It demonstrates that a simple solution to the problem of learning to traverse long action chains

[1] Barry (2000b) has shown that XCS is able to learn the optimal solution to the GREF-1 (Riolo, 1987) environment more rapidly than CFSC.

within simple progressive corridor environments exists. It then shows that the addition of hierarchical control will allow this solution to be applied to a more complex environment. This expansion represents the first application of hierarchy within XCS, and the methods are contrasted with those of Dorigo and Schnepf (1993) and Dorigo and Colombetti (1994) with ALECSYS. The approaches are also related to two methods for hierarchical learning within Reinforcement Learning, demonstrating that techniques within Reinforcement Learning can be utilised beneficially within LCS.

2 LOCALISATION OF REWARD

In resolving the difficulties involved with the learning of Maze 14 an understanding of the core problem of the learning task is essential. Section 1 highlighted the inability of exploration to move the animat controlled by XCS from the early states towards the later states in this environment due to the properties of the environment itself. As a result, there is little opportunity to identify the reward state or to feed the payoff back to earlier classifiers. Equally, the inability to distinguish between classifiers in earlier states due to non-availability of reward information prevents consolidation of reward feedback as a result of learning within exploitation.

This dilema can be partially solved if the LCS was able to introduce intermediate rewards in addition to that provided by the environment. Consider a state s_i which is i steps from the start state s_0. If a classifier leading to this state from s_{i-1} received a fixed 'internal' reward R_I, this reward could be fed back to preceeding classifiers. Thus, the LCS would be able to establish classifiers leading to state s_i even though the ultimate goal state had not yet been encountered. Now consider a set of states s such that $s \subset S \wedge \forall s_i \in S \cdot \neg\exists s_j \in S \cdot j = i + 1$ where $S = \{s_0 \ldots s_{n-1}\}$. If each of the states within s was a state providing an internal reward, the states within s represent a chain of intermediate goals towards which XCS can learn a route.

The provision of these internal reward states is not in itself sufficient to enable XCS to find a path to the solution. XCS must, in addition, be able to identify which of the internal reward states to move towards next, and equally must be able to decide not to re-visit an internal state that has already been visited. It is possible to consider an XCS implementation in which the state space is subdivided and for each sub-division one of the internal 'sub-goal' states is advocated and the internal reward is paid out when the Animat controlled by XCS enters this state. For the situation where there is only ever a single sub-goal per environment subdivision the Optimality Hypothesis (Kovacs, 1996) implies that XCS will learn the optimal state × action × payoff mapping for each environmental subdivision. The problem of learning a route from the start state to the reward state is thus decomposed into the problem of moving from one internal reward state to another internal reward state.

Hypothesis 1

Using a prior identification of internal goal states and subdivision of the state-space in relation to the goal states, XCS is able to learn the optimum state × action × payoff mapping for each subdivision, and given a mechanism to determine the sequence of internal goals an optimum path to a global goal can be constructed.

Limiting the environment to a single sub-goal per environmental subdivision limits this mechanism to unidirectional corridor environments. Clearly this is an undesirable limitation. The limitation may be overcome by identifying more than one goal state within each subdivision. Providing the conditions for the classifiers within the XCS are constructed to identify both the current goal and the current local state, it is hypothesised that the Optimality Hypothesis can be extended so that the populations covering each state-space decomposition will be able to identify its optimal state × sub-goal × action × payoff mapping for that part of the state-space.

Hypothesis 2

Where more than one sub-goal state exists within a state-space subdivision and the desired sub-goal is made available through the input mechanism, XCS is able to learn the optimum state × sub-goal × action × payoff mapping for each subdivision, and given a mechanism to determine the sequence of internal goals a sequence of optimum local routes to a global goal can be constructed.

The mechanism for the selection of the current 'goal' states or the relevant XCS sub-population has not been discussed thus far. For this investigation it is proposed that the method of requiring the user to identify the state subdivisions and their "terminal states" used within many Reinforcement Learning approaches (such as Diettrich, 2000; Parr and Russell, 1997) is adopted. As such, the structures used are *fixed* rather than *emergent*. Given this input it is hypothesised that an additional high-level XCS can be added that operates over the space of internal states, treating the lower XCS sub-populations as "macro-actions" that move from the current state to the chosen sub-goal state. Given the current input (which will be one of the sub-goal states) the high-level XCS will select a new sub-goal state and a low-level XCS population to reach that sub-goal. Upon reaching the subgoal state this lower-level XCS will be rewarded the internal reward and will hand control back to the high-level XCS. When the environmental reward state is reached, the high-level XCS will receive the environment reward and through the normal payoff mechanism it is hypothesised that it will learn the optimal state × next sub-goal × payoff mapping.

Hypothesis 3

An XCS can be employed to learn the optimum sequence of sub-goals from a pre-defined set of sub-goal states to reach a reward state within an FSW by the invocation of low-level XCS populations each mapping a unique sub-division of the state-space.

This fixed hierarchical structure using pre-defined sub-goal states does not represent a fully emergent solution.

However, the demonstration of the ability of XCS to operate within these structures and retain the advantages inherent within XCS at each level will pave the way towards further work leading to truly emergent hierarchical XCS formulations. In addition, a demonstration of the validity of the hypotheses provides new solutions to the problem of learning within environments requiring long action chains, and opens the possibility of re-using learnt mappings within more than one area of the state space.

3 EXPERIMENTAL APPROACH

The XCS implementation used within this work is XCSC (Barry, 2000b). In the experimental investigation of the hypotheses the operation of the base implementation will be changed as little as possible to achieve the required structured XCS architectures. In order to maintain comparability with previous work on action chain length (Barry, 2000b, 2001) the parameterisation used within these experiments is as follows: N=400, p_1=10.0, ε_1=0.01, f_1=0.01, R=1000, γ=0.71, β=0.2, ε_0=0.01, α=0.1, θ=25, χ=0.8, μ=0.04, P(#)=0.33, s=20 (see Kovacs (1996) for a parameter glossary). The Finite State World (Riolo, 1987; Barry, 1999) environment used is depicted in figure 1.

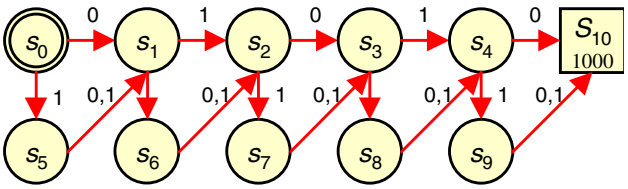

Figure 1 - An extensible corridor test environment suitable for testing action-chain learning within XCS.

This environment has the following useful features:

- It can be trivially extended by small or large increments as longer test action chains are required;
- It includes a choice of route at each state so that the ability of XCS to decide the optimal route as the action chain increases can be determined;
- The sub-optimal route does not prevent progress towards the reward state;
- The optimal route is always re-joined to limit the penalty of a sub-optimal choice;
- The stable payoff received for a sub-optimal choice will always be equivalent to the γ discount of the payoff received for the optimal choice;
- The alternation of actions prevents generalization from prematurely producing very general classifiers that cover much of the optimal path to reward;
- The small number of separate actions limits exploration complexity.
- The environment can be sub-divided into sections each of which has a single identifiable sub-goal state.

The unsuitability of some of the 'standard' XCS performance measures for multiple-step environments has been rehearsed elsewhere (Barry, 1999, 2000b). The System Relative Error (Barry, 1999) is therefore adopted as the standard performance metric and the coverage table (Barry 2000a) is used to identify the level of convergence on the optimal classifier for each action set. As hierarchical structures are introduced the validity of some of the performance measures previously used within XCS become strained. Where appropriate, therefore, these measures are then applied locally and reported separately for each sub-population, and other means are introduced to provide a measure of global performance. Such changes are identified alongside the experimental investigations as they are required.

4 SUB-DIVIDING THE POPULATION

In order to investigate Hypothesis 1 a simple structuring of the population space within XCS was devised. The approach taken is based on the methods used within HQ learning (Wiering and Schmidhuber, 1996), although greatly simplified. The developed XCS formulation is therefore known as a *Simple* H-XCS (SH-XCS),. The standard XCS implementation was modified so that an array of populations is maintained rather than a single population and a variable is added to reference the current population. The environmental interface is modified so that a set of states from the environment can be identified as internal reward states (to become the sub-goals) and operations to allow XCS to detect when an internal state has been reached are provided. The environment is also modified to allow the provision of an internal reward value for reaching an internal state, again with operations that allow XCS to obtain that reward value.

The operation of XCSC is modified so that at the start of each trial the first sub-goal is identified and the current population is set to the first population. XCS then runs as normal within this population until the environment identifies that an internal sub-goal or global goal has been reached. On reaching an internal sub-goal, the state is compared with the desired internal goal and if it is the same the internal reward is provided, the current population variable is moved on to the next sub-population, and the next sub-goal is identified. Upon reaching the global goal the same internal reward is provided to the current sub-population and the global reward is discarded. Thus far the modifications can all be related to features found within a HQ implementation. The simplification comes in the selection of the current sub-goal. Within HQ learning this is performed by an additional HQ table associated with each Q-table - the HQ table uses the current global goal to select a local sub-goal for the local table. The modifications to XCS are concerned with verifying Hypothesis 1, and this does not require a higher-level choice of sub-goal. Therefore the choice of the next sub-goal is deterministic and is simply the next sub-goal in the available sub-goal list. Thus each sub-population learns the optimal path to one sub-goal. SH-XCS was tested by running both XCS and SH-XCS starting with the same random seed in the length-10 (20

state) version of the test environment. It was found that SH-XCS with a single sub-population and a single sub-goal at the global goal produced identical performance plots to XCS for all the standard results collected, confirming that the modifications had not changed the normal operation of XCS.

4.1 INVESTIGATING HYPOTHESIS 1

SH-XCS was now applied to the length-10 environment using two sub-populations. The sub-goal states were s_5 and s_{20},[2] with the goal state also s_{20}. The condition size was set to 6 bits. A total population size of 800 was used, divided equally between the two sub-populations. The internal reward value was set to 600, a value chosen because of the known reduction in confusion that its discounted values cause when the main reward is 1000 (Barry, 1999b) (although not an issue at this stage). Ten runs of SH-XCS were performed with up to 100 iterations per trial. The performance of the whole learning system was captured using the System Relative Error metric. The population size measure was modified so that it captured the size of each sub-population rather than providing a single result. This gives a means of tracking the comparative rate of learning in terms of the focus of the sub-populations on their optimal sub-populations [O].

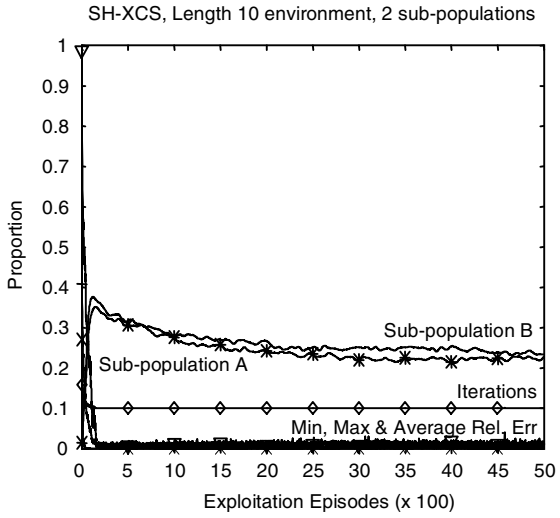

Figure 2 - The performance of the sub-populations of SH-XCS within a length 10 two-choice progressive corridor FSW

Figure 2 pictures the performance of SH-XCS in this experiment. SH-XCS rapidly converged on solutions for each of the sub-populations, and the System Relative Error of the two sub-populations was rapidly eliminated. The population curves show that each population has converged onto a solution and at 5000 exploitation episodes continue to consolidate on their respective [O]. A comparison of these results with those obtained from XCS in the same environment indicated that SH-XCS learns the optimal number of steps in which to traverse this environment in the same length of time as that taken by XCS within an equivalent environment of length 5. Thus, the two sub-populations operating within their own length 5 portion of the length 10 environment are able to establish a solution to their state-space in the same time as a single XCS in an equivalent length 5 environment. It is important to note that the environments tackled by XCS in the length 5 test and the environments tackled by XCS in the two length 5 sub-divisions of the state space within this test are not the same. The state encoding for the length 10 environment was not changed when it was sub-divided so that the advantages that might be gained by a reduced input space (Dietrich, 2000) result from the reduced size of the search space only. Thus, the generalisation task undertaken in each of the sub-populations was different and leads to different [O] within each sub-population.

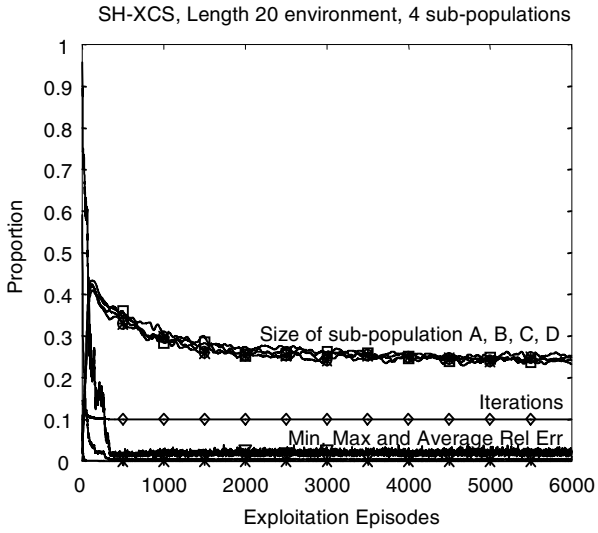

Figure 3 - The performance of four sub-populations of SH-XCS within a length 20 two-choice progressive corridor FSW.

The test environment was now extended to length 20, a length that Barry (2000b) demonstrated could not be adequately learnt using the standard XCS. Four sub-populations were provided, each of length 5 as in the previous experiment. Sub-goal states were 5, 10, 15 and 40, with 40 also providing the goal state. The parameterisation was kept constant apart from the total population size, which was increased to 1600 (divided between each sub-population). Figure 3 gives the averaged performance within the first 6000 exploitation trials from ten runs of 15000 exploitation trials. The similarity of the results presented with those in the length 10 environment is striking. Even though the bit length of the message was increased from six bits to seven and the distance between the decimal value of the messages from the 'optimal route' states and the messages from the 'sub-optimal route' states has increased the learning rate within each sub-population has changed little. This bears out Wilson's hypothesis (Wilson, 1998) that the difficulty experienced by XCS in finding [O] scales with

[2] The term 'Length n' refers to the number of states on the optimal route. There are also an equal number of states on the sub-optimal route (fig 1).

generalisation difficulty rather than bit length. This is particularly relevant for the development of hierarchical approaches using XCS, since the requirement to physically reduce the input size for each sub-population in order to see beneficial performance improvements within Diettrich's MaxQ approach (Diettrich, 2000) may not apply to hierarchical XCS solutions in the same way. Other experiments (see Barry, 2000b) revealed that performance improvements can in fact be gained by utilising input optimisations. This is logical, since a reduction in the message size should require a smaller population to learn the generalisations and should therefore produce performance improvements.

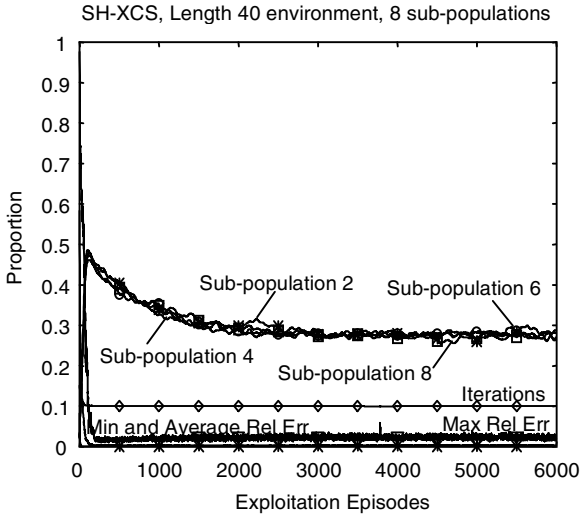

Figure 4 - The performance of eight sub-populations of SH-XCS within a length 40 two-choice progressive corridor FSW.

An analysis of the coverage tables produced from these runs revealed that each sub-population had learnt and proliferated [O] and that [O] was dominant to approximately the same degree as SH-XCS with two sub-populations within the length 10 environment. This is unsurprising, since the learning problem for each sub-population has only been modified in terms of the generalisations to be formed and not in terms of the size or structure of the underlying state-space. Given this finding, the SH-XCS approach would be expected to continue to scale to larger environments penalised only slightly by the additional number of bits required to encode the enlarged state space. To evaluate this claim the environment was increased once more to provide a length 40 action chain to the reward. Eight sub-populations were provided and the sub-goal states were 5, 10, 15, 20, 25, 30, 35, and 80. The condition size was 8 bits and the total population size was 3200.

As expected, the SH-XCS implementation continued to learn rapidly how to traverse this extended environment (figure 4), and once more each sub-population developed a dominant [O]. Given these results, it can be concluded that the use of a prior identification of internal goal states and the subdivision of the state-space in relation to the goal states allowed XCS to learn the optimum state × action × payoff mapping for each subdivision of the state space. Similarly it is concluded that this capability can be used to construct a set of an optimal local paths to a global goal. Therefore Hypothesis 1 is upheld.

4.2 HIERARCHICAL CONTROL

Whilst Hypothesis 2 could be investigated by extending SH-XCS to provide each population with a deterministic sub-goal identification mechanism, it was decided to examine hypotheses 2 and 3 using one mechanism. The Feudal Q-Learning approach to reinforcement learning (Dayan and Hinton, 1993) is a simple approach to hierarchy construction that requires a pre-identified sub-division of the state space into small Q-tables and a pre-selected hierarchy of Q-tables. A Q-table at level *n* in the hierarchy would learn the optimal choice of Q-table from the sub-division of Q-tables at the level *n + 1*. Thus, at the top of the hierarchy a single Q-table would exist, and an inverted tree of successive levels of hierarchy would be constructed until the lowest level of Q-tables operated over a choice of actions on the environment rather than a choice of Q-tables. Each Q-table in levels above the lowest acted like a feudal Lord - they had oversight of a distinct sub-space within the environment and they decided the sub-goal that a selected lower-level Q-table would have to seek to achieve.

The Feudal Hierarchy approach is very close to the form of hierarchical control that hypotheses 2 and 3 pre-suppose. It is therefore appropriate to seek to apply this form of hierarchy within XCS as a natural extension to the previous work with SH-XCS. Rather than implement this "*Feudal XCS*" as a hierarchy of populations a simpler implementation strategy was chosen. It was recognised that if an upper level *n* XCS selects a sub-population and chooses a sub-goal at the next level down (*n-1*) then the set of lower populations and their sub-goals can be seen as the environment that the level *n* XCS is operating upon. If the level *n-1* sub-populations were themselves instances of XCS, then the choice of a sub-population can be viewed as *invoking* a lower XCS to run an episode that seeks to reach the specified sub-goal.

The standard XCSC was therefore modified so that the environment for any level XCS above the base level was an XCS instance. To invoke the lower XCS an upper XCS would write the selected sub-goal into the environment of the lower XCS and then invoke a trial of the lower XCS. Whilst all levels of the Feudal Hierarchy are given input from the current environmental state, all levels apart from the uppermost will also have the current sub-goal state for their level identified in their input message. It was recognised that a full specification of the sub-goal within the message would double the message size of the lower XCS and that only a small number of states covered by the lower level XCS would be used as potential sub-goals. Therefore the sub-goals within each environment subdivision were identified within a user-supplied table and the sub-goal choice and message are constructed from the index into that table.

Upon invocation, the lower level XCS will use its population to find the best route to the sub-goal selected by the upper XCS. If the current state is outside the area covered by the selected lower XCS then it will immediately return without any further action (or payoff) so that the discounted payoff mechanism identifies the selection of that sub-population as a "*null action*". Otherwise the XCS uses its population to identify (and learn) the optimal route to the chosen sub-goal. During operation of the lower XCS any action that would cause movement out of the state subdivision covered by that XCS is prevented so that each state space decomposition is treated as though it were the only state-space for that XCS. If the sub-goal is achieved within the number of steps allowed for a trial of the XCS an internal reward value is given to the XCS. If the sub-goal is not achieved no reward (or penalty) is given - the temporal difference update will identify the route as sub-optimal without penalty. At the end of a trial control is handed back to the upper level XCS without reward. The uppermost XCS is the only XCS to receive environmental reward, and will use temporal difference to learn the optimal choice of sub-populations and sub-goals from this payoff. Each trial of XCS at any level is an unaltered XCS trial, including normal induction algorithms. However, the explore-exploit choice is specified by the uppermost XCS.

The capture of integrated reports even for the simple two-level hierarchies used within this investigation is problematic - the learning rates of the two levels are different and invocation of each sub-population will occur at different rates within any non-trivial environment. Therefore each sub-population produces separate reports and these results are gathered for presentation as appropriate to the experiment.

4.2.1 Feudal XCS in a unidirectional Environment

The Feudal XCS was created and after appropriate testing was applied to the same length ten environment used within section 4.1 so that comparative performance data could be gained. The length 10 environment was subdivided into two length 5 environments to correspond to the decomposition within section 4.1. State 5 was designated as the sub-goal for the first subdivision and the reward state, state 20, was the sub-goal for the second. Since the aim of the Feudal XCS is to allow an upper level XCS to prescribe not only the sub-population to use but also the sub-goal to move towards, for each sub-division two sub-goals were specified although they both referenced the same sub-goal state. The message for the top-level XCS consisted of the current state, with its output specifying the lower level XCS to use (1 bit) and the sub-goal to select (1 bit). The message for the lower level XCS instances consisted of the current state and the sub-goal specified by the upper level (1 bit). The action consisted of the direct environmental action (1 bit). Experiments demonstrated that Feudal XCS was able to learn the optimal selection of the lower XCS within this environment, and so the experiment was extended to a length 20 environment requiring four lower level XCS instances. As figure 5 illustrates, Feudal XCS was able to concurrently learn the optimal choice of sub-population and the optimal route to the sub-goal state. The dominance of [O] was good in each XCS population.

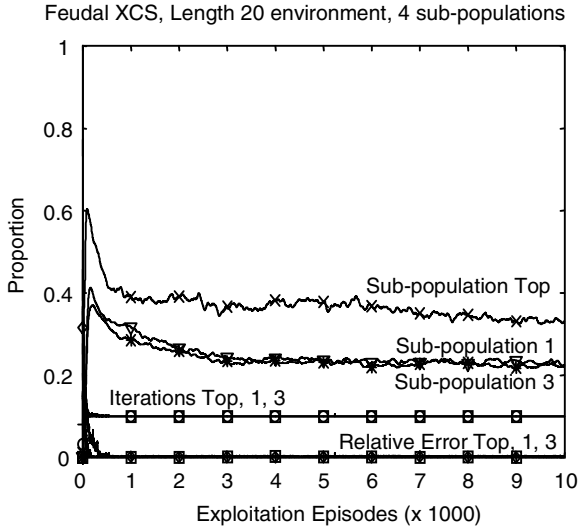

Figure 5 – Feudal XCS in length 20 unidirectional environment

4.2.2 Feudal XCS in a two subgoal environment

Having demonstrated that Feudal XCS is able to select the optimal sub-XCS and then find the optimal local pathway attention was turned to the ability of Feudal XCS to operate within an environment where each state-space sub-division identified two sub-goals at different locations. A suitable environment is pictured in figure 6.

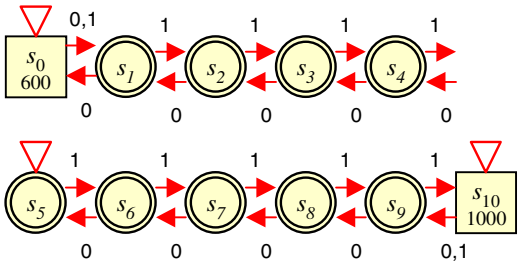

Figure 6 - A corridor environment with two sub-goals in each of two state-space sub-divisions

The state-space was divided into two, with states s_0 to s_5 within the first sub-division and states s_5 to s_{10} within the other. The sub-goals identified were states s_0 and s_5 in the first subdivision and states s_5 and s_{10} within the second. In this environment the upper XCS must learn both the optimal sub-goal and which lower level XCS to select from in any state, and the order of choice of the lower XCS instances and sub-goals required in order to maximise payoff from the two payoff sources. Through a number of pre-experimental runs it was found that the optimal population size for both the upper and lower XCS instances was 400. The condition size for the top population as set to four bits, with a two bit action (bit 0 =

sub-population bit 1 = sub-goal). The condition size of the bottom populations was set to five bits - four for the current state and one for the desired sub-goal. The action size remained 1 bit for the selection of the environmental action. An examination of the performance of the sub-populations under exploration also revealed that the limit of 50 steps within a population led to the sub-populations on occasions not achieving their sub-goal. This had the effect of introducing fluctuating payoff to the upper XCS, preventing a full reduction in System Relative Error. This was rectified by allowing the lower XCS to continue until a subgoal state was discovered. After these modifications were made the experiment was run.

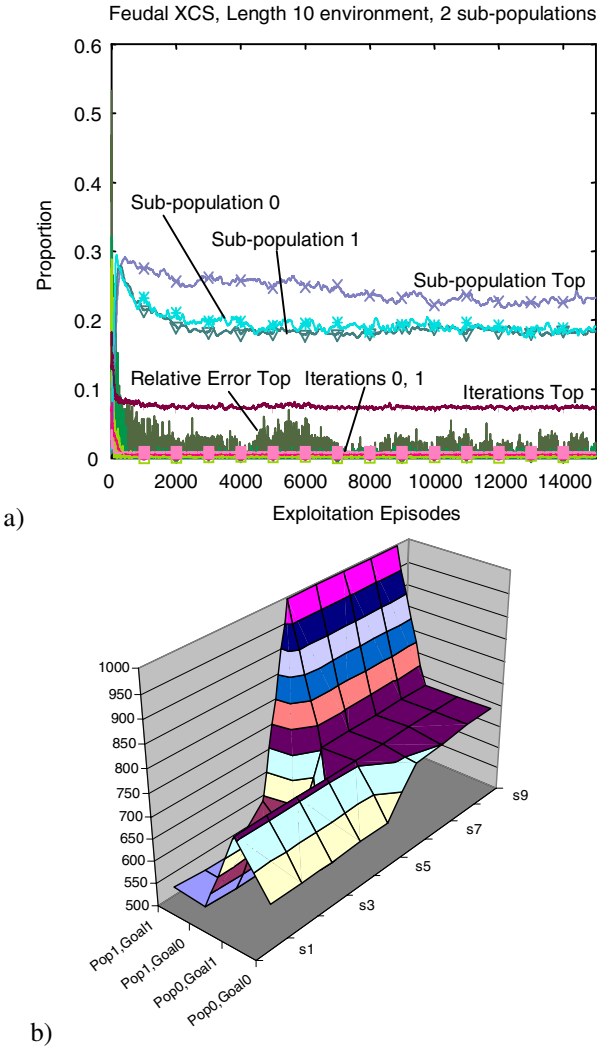

Figure 7 - a) The performance for Feudal XCS with two sub-populations in a two-goal length 11 corridor environment, and b) the coverage graph for the top-level population.

In Figure 7a the System Relative Error of the high-level XCS did not reduce as much as expected. It was hypothesised that this was due to the uneven nature of exploration - states s_0, s_5, and s_9 would be explored more regularly than the other states and similar problems had been seen in previous experiments (Barry, 2000a). This hypothesis was verified by reducing the start states to s_0, s_3, s_5, s_7, and s_9. It was then found that the dominance of [O] was normal and the System Relative Error was reduced to expected values. The dominance of [O] for the low-level XCS populations was high in all runs, demonstrating the ability of Feudal XCS to identify the optimal local state × sub-goal × action × payoff mappings, empirically verifying the first section of hypothesis 2. The second section of hypothesis 2 suggested that given a suitable policy these sub-populations could be used to provide a sequence of optimal local routes to achieve a global goal. This is demonstrated by the iterations plot for the top-level XCS in figure 7a. This plot reveals that the Feudal sub-population is able to achieve a global goal using the optimum one or two sub-population invocations (the line is plotted so that 0.1 on the scale represents the optimal two steps for the longest path).

A consideration of figure 7b reveals that the high-level XCS was able to identify the optimum pathway, using the lower level sub-goals and sub-populations, that will achieve the highest payoff from the environment. This demonstrates that the mapping created is the optimal global mapping of state × sub-population × sub-goal × payoff and thus hypothesis 3 is also upheld.

Whilst the Feudal XCS did acquire the capability to select between global payoffs, it should be noted that the global payoff chosen by XCS will not necessarily be that chosen by the normal XCS. For example, in the environment used for these experiments XCS will select a route to the state s_0 that provides the reward of 600 when starting in states s_1 to s_4 and the route to the state s_{10} that provides the reward of 1000 when starting in states s_5 to s_9. In Feudal XCS the reward of 1000 is a maximum of two 'macro-steps' away from any starting location, and therefore XCS will always prefer the sequence of sub-goals leading to s_{10}. Thus, the high level XCS population plans over sub-goals rather than individual states. As McGovern and Sutton (1998) note, this form of hierarchical approach produces routes to reward states that are *optimal at the level of planning*.

5 DISCUSSION

Previous work with Structured LCS is explored in more detail within Barry (2000b), to which the interested reader is directed regarding other related work. Some of this previous work is particularly close to the work presented here, and is worthy of consideration at this point. Booker (1982) used multiple instances of his GOFER LCS implementation to differentiate between input and output mappings and enable the LCS to learn internal associations between input and output. This represents a different aim to that of the Feudal XCS which focuses on learning to plan over concurrently learnt subgoals and competences. Bull and Fogarty (1993) used a number of classifier populations that could switch each other on or off by messages to a shared message list. These LCS populations were stimulus-response systems, although

learning a long-term behaviour, and this work therefore has much in common with the work of Dorigo.

The main body of previous investigation into hierarchical forms of LCS was performed by Dorigo and colleagues (e.g. Dorigo and Schnepf, 1993; Dorigo and Colombetti, 1994). Using ALECSYS they created fixed control hierarchies. Their work was characterised by the dependency upon direct environmental feedback for the reward of switching decisions made by the upper level LCS. Their bottom-up hierarchical approach required input to be divided between the low level populations. Each decided whether to propose an action, and the top-level LCS chose between the actions. In an alternative top-down approach a state memory was used to identify the current goal. Each lower level LCS learnt to use the state memory to identify which LCS should operate and a co-ordinator LCS learnt to control this memory switch. Although the learning environments were multiple-step environments, a regular payoff for each action was provided and training was performed separately.

In contrast, Feudal XCS is designed to learn within delayed-reward environments - the purpose of Feudal XCS is the decomposition of action sequences into smaller units and the localisation of reward within those units. Secondly, the Feudal XCS selects lower level capabilities based on identified sub-goals, and uses these to plan at a higher level. Whilst ALECSYS did select between behavioural competences, it did not use the competences to identify sub-goals that established a route to a rewarding state. Finally, the Feudal XCS maintains all the capabilities of XCS to acquire accurate and optimally general mappings of each state-space partition and sub-goal space, which is not possible in ALECSYS.

Much further work remains to be done to assess the scalability and wider applicability of this approach. It must be applied to larger numbers of sub-divisions and scaled to operate with more than one level of decomposition. In particular, exploration of the potential for autonomous identification of subgoal states would lead to a truly emergent hierarchical approach. However, these results do provide encouragement and expand upon the available research results for hierarchical LCS formulations.

References

Barry, A.M. (1999), Aliasing in XCS and the Consecutive State Problem: 1 - Problems. In Banzhaf et al, 1999.

Barry, A.M. (2000a), Specifying action persistence in XCS. In Whiteley et al, 2000.

Barry, A.M. (2000b), XCS performance and population structure in multiple-step environments, PhD Thesis, Queens University Belfast.

Barry, A.M. (2001), The stability of long action chains in XCS, to be published in Bull, L., Lanzi, P-L (eds), The Journal of Soft Computing, Sept 2001.

Banzhaf, W. et al. (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*. Morgan Kaufmann: San Francisco, CA, 1999.

Booker, L. B. (1982), *Intelligent Behaviour as an Adaptation to the Task Environment*, Ph.D. Dissertation, The University of Michigan.

Bull, L., Fogarty, T. C., (1993), Co-evolving Communicating Classifier Systems for Tracking, in Albrecht, R.F. et al (eds.), *Proc. Intl. Conf. on Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag.

Butz, M. V., Stolzmann, W., Goldberg, D. E., (2000), Introducing a Genetic Generalisation Pressure to the Anticipatory Classifier System Part 2: Performance Analysis, In Whiteley et al, 2000.

Cliff, D., Ross, S., (1994), Adding Temporary Memory to ZCS, *Adaptive Behaviour*, 3(2), 101-150.

Dayan, P., Hinton, G. E. (1993), Feudal reinforcement learning, in Hanson, S. J. et al (eds.), *Neural Information Processing Systems 5*, Morgan Kaufmann.

Dietterich, T. G. (2000), *An Overview of MaxQ Reinforcement Learning,* Technical Report, Computer Science Department, University of Oregon.

Dorigo, M., Colombetti, M., (1994), Robot shaping: Developing autonomous agents through learning, *Artificial Intelligence*, 71 (2), 321-370, Elsevier Science.

Dorigo, M., Schnepf, U. (1993), Genetics-based Machine Learning and Behavior-Based Robotics: a new synthesis., *IEEE Trans. Systems, Man, and Cybernetics*, 23(1).

Kovacs, T., (1996), Evolving optimal populations with XCS classifier systems. Tech. Rep. CSR-96-17, School of Computer Science, University of Birmingham, UK.

Lanzi, P.L., (1997), Solving problems in partially observable environments with classifier systems, Tech. Rep. N.97.45, , Politecnico do Milano, IT.

M^{c}Govern, A., Sutton, R. S. (1998), *Macro-Actions in Reinforcement Learning: An Empirical Analysis*, Technical Report 98-70, Computer Science Department, University of Massachusetts, Amherst.

Parr, R., Russell, S. (1998), Reinforcement Learning with Hierarchies of Machines, in *Advances in Neural Information Processing Systems*, 10, MIT Press.

Riolo, R.L. (1987), Bucket Brigade performance: I. Long sequences of classifiers, in *Proc. Second Intl. Conf. on Genetic Algorithms and their Applications*, 184-195.

Whitely, D., Goldberg, D. E, Cantú-Paz, E., Spector, L., Parmee, I., Beyer, H-G., (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Morgan Kaufmann.

Wiering, M., Schmidhuber, J. (1996), HQ-Learning: Discovering Markovian Sub-Goals for Non-Markovian Reinforcement Learning, Technical Report IDSIA-95-96.

Wilson, S.W. (1998), Generalization in the XCS Classifier System, in *Proc. 3rd Ann. Genetic Prog. Conf.*

Classifier systems, endogenous fitness, and delayed rewards: A preliminary investigation

Lashon B. Booker
The MITRE Corporation
1820 Dolley Madison Blvd
McLean, VA 22102-3481
booker@mitre.org

Abstract

Previous work has shown the potential advantages of using endogenous fitness schemes in classifier systems. The basic idea behind endogenous fitness is to reinforce successful system performance with "resources" that rules need in order to reproduce. Instead of storing explicit quantitative estimates of performance, each rule has one or more reservoirs that are used to store resources. When enough resources have been accumulated, a rule utilizes some of its resources to reproduce and the reservoir level is reduced accordingly. This paper extends this concept to accommodate environments having delayed rewards. Reinforcement learning techniques for solving average-reward Markovian decision processes are combined with a simple endogenous fitness scheme in a classifier system. We describe initial tests of this approach on state-space search problems used in previous classifier system studies.

1 Introduction

Classifier system implementations have traditionally used explicit measures of utility — such as predicted payoff, accuracy, payoff-derived strength, etc. — to quantify the utility and fitness of classifier rules. This research is investigating classifier systems that determine the utility and fitness of rules endogenously, without computing explicit estimates.

The basic idea behind endogenous fitness schemes is straightforward. The rules advocating an action on a given time step (i.e., the action set) take all of the credit for whatever reinforcement is received. Each reinforcement event leads to the distribution of some nominal resources among those rules, and the acquired resources accumulate over time in internal reservoirs. Following the way resources are used in Echo [6], the endogenous fitness scheme allows rules to reproduce only when they have accumulated resources in excess of some threshold amount. When enough resources have been accumulated, a rule utilizes some of its resources to reproduce and the reservoir level is reduced accordingly. A distinction is made between reinforcement events that are better than "average" and events that are worse than average. The cumulative difference in resources a rule has received for these two outcomes is used to determine eligibility for reproduction. The relative amount of resources received for these two outcomes is the basis for a prediction about the outcome (i.e., reinforcement better or worse than "average") expected whenever a rule is active.

Previous work [4] described one implementation of this idea. That classifier system has a fairly conventional design, borrowing elements from previous work on GOFER [2, 3] and XCS [11]. The population of classifiers has a fixed size $\mathcal{N}$, each classifier having a single condition on the left side and a single action on the right side. Each classifier ξ has two associated reservoirs: the $\Delta_+(\xi)$ reservoir that stores resources obtained from "better than average" reinforcement events, and the $\Delta_-(\xi)$ reservoir that stores resources obtained from "worse than average" reinforcement events. The only other parameters stored with each classifier are: age $\alpha(\xi)$, which is used in the procedure for deleting classifiers; an estimate $\pi(\xi)$ of the average reward available when ξ is included in the match set; and, a counter $\nu(\xi)$ that records the number of times ξ has been included in the match set on an "explore" trial. Classifiers are eligible to reproduce when the difference $|\Delta_+(\xi) - \Delta_-(\xi)|$ is larger than some threshold. Empirical performance of the endogenous fitness scheme implemented in this way were encouraging. The system performs as well as utility-

based classifier systems such as XCS [11] on the multiplexor problem.

One of the important research issues not addressed by this previous work on endogenous fitness in classifier systems is how to solve multi-step reinforcement learning problems involving sequences of actions and delayed rewards. In this paper we describe work in progress that is extending the endogenous fitness scheme to handle such problems.

We begin with a brief discussion of our latest approach to implementing endogenous fitness in classifier systems. That discussion is followed by a description of work in progress that integrates average-reward reinforcement learning techniques into the endogenous fitness paradigm.

2 Implementing Endogenous Fitness In Classifier Systems

In the current implementation there are also no explicit, individual performance estimates associated with classifiers. Each classifier ξ has the two associated reservoirs $\Delta_+(\xi)$ and $\Delta_-(\xi)$. The reservoirs are initialized to be empty and the initial classifiers are generated at random. The following additional parameters are stored with each classifier: an estimate $\pi(\xi)$ of the average reward available when ξ is included in the match set; estimates $\delta_+(\xi)$ and $\delta_-(\xi)$ of the average reward available when the best and worst actions in $\mathbf{M}$ are selected (as determined by the performance system); a counter $\alpha(\xi)$ that records the number of times ξ has been included in the action set; and, an estimate $\omega(\xi)$ of the proportion of times the reward available exceeds $\pi(\xi)$ when ξ is in the action set. These parameters are used to help characterize the flow of resources in a match set. Here we provide a brief summary of the key details, focusing primarily on those that differ significantly from the description given in [4].

2.1 Performance System

The system performance cycle is fairly routine. For each input message i, the system first determines the set of classifiers $\mathbf{M}$ eligible to classify the message. Matching classifiers are always included in $\mathbf{M}$. Following the procedures in GOFER, if there are fewer than $\mathcal{N}_m$ matching classifiers available, classifiers with the highest partial match scores are deterministically selected to fill out $\mathbf{M}$. We use the simple partial match score

$$\mu(\xi, i) = \begin{cases} s + l & \text{if } \xi \text{ matches the message } i \\ l - n & \text{otherwise} \end{cases}$$

where l is the length of the input condition in ξ, s is the specificity, and n is the number of positions where the condition doesn't match the message.

For each action a represented in the match set $\mathbf{M}$, the system computes an *action mandate* that captures the system's knowledge about the likelihood of a "better than average" outcome if action a is chosen. Each classifier ξ in $\mathbf{M}$ computes the value

$$\lambda(\xi) = 2 \mid \omega(\xi) - 0.5 \mid$$

as the mandate for its action. Note that $\lambda(\xi)$ is 1 whenever the outcome associated with ξ is consistently better or worse than average ($\omega(\xi) = 0$ or 1) and 0 when the outcome is random ($\omega(\xi) = 0.5$). When $\omega(\xi) \geq 0.5$, this contribution from each rule is added to an *action selection array*. When $\omega(\xi) < 0.5$, this contribution from each rule is subtracted. The rationale for this approach is to give a higher net weight to those actions that, based on previous experience, have the highest likelihood of being followed by a "better than average" outcome.

As in XCS, the information in the action selection array is used to determine which action is selected. The members of $\mathbf{M}$ that agree with the selected action constitute the *action set* $\mathbf{A}$. The system then sends that action to the effectors, and the environment may respond with reinforcement.

2.2 Reinforcement

On every time step, parameters of the classifiers in $\mathbf{M}$ are adjusted and some amount of resource $\mathbf{R} > 0$ is made available to the classifiers in $\mathbf{A}$. Competition for this resource is the primary mode of interaction among the rules in the population. The following sequence of steps is used to determine how the resource is distributed:

- The $\pi(\xi)$ parameter is revised for all classifiers in $\mathbf{M}$ using the simple update rule

  $$\pi_t(\xi) = \begin{cases} \frac{(\pi_{t-1}(\xi)\nu_{t-1}(\xi)) + \mathcal{R}}{\nu_t(\xi)} & \text{if } \nu_t(\xi) \neq \nu_{t-1}(\xi) \\ \pi_{t-1}(\xi) & \text{otherwise} \end{cases}$$

  where $\mathcal{R}$ is the reward received, $\nu(\xi)$ is a counter that records the number of times ξ has been included in the match set, $\pi_0(\xi) = 0$, and $\nu_0(\xi) = 0$. If the action a is the best (or worst) option available, then a similar update is made to $\delta_+(\xi)$ (or $\delta_-(\xi)$).

- The members of $\mathbf{M}$ collectively estimate the average reward Π for the current state as the central

tendency of the values $\pi(\xi)$ in $\mathbf{M}$. Since $\mathbf{M}$ will often include overly general rules with inaccurate values, it is helpful to take some steps to avoid having this estimate contaminated. Order statistics can provide a robust estimate of the central tendency. We use a conservative boxplot criterion [8] to identify outlying values and exclude them from the computation. The boxplot criterion computes the median $\tilde{x}$ of the data values, the lower quartile q_1, and the upper quartile q_3. Any value that lies $3(q_3 - q_1)$ above the upper quartile or below the lower quartile is labeled as an outlier. The trimean estimator [1], given by

$$\hat{x} = \frac{q_1 + 2\tilde{x} + q_3}{4}$$

is used to obtain a simple and reasonably robust estimate of the central tendency Π. While there are many other ways to compute the central tendency that give adequate results, the methods using order statistics have given the best results so far.

In an analogous manner, there is a collective determination of the central tendencies $\hat{\delta}_+$ and $\hat{\delta}_-$ of the parameters $\delta_+(\xi)$ and $\delta_-(\xi)$ respectively.

- The resource $\mathbf{R}$ available on each time step is scaled to reflect the size of the reward $\mathcal{R}$ relative to what is expected in $\mathbf{M}$. It is sufficient to use a simple linear scaling given by

$$\mathbf{R} = \bar{\mathbf{R}} \left(1.0 + \frac{\mathcal{R} - \hat{\delta}_-}{\hat{\delta}_+ - \hat{\delta}_-} \right)$$

where $\bar{\mathbf{R}}$ is a system parameter indicating the minimum amount of resource made available on each time step. Given two classifiers that are consistently associated with above average rewards, this procedure gives a modest selective advantage to the classifier that is best from a payoff standpoint.

- Each classifier in $\mathbf{A}$ receives a share of the resource given by

$$\rho(\xi) = \left(\frac{\lambda(\xi)\mathbf{H}(\xi)}{\sum_{\xi \in \mathbf{A}} \lambda(\xi)\mathbf{H}(\xi)} \right) \mathbf{R}$$

where $\mathbf{H}(\xi)$ is a hypergeometric probability that helps bias the distribution of resources to favor sets of rules that efficiently cover all input messages. This computation is strongly related to the familiar fitness-sharing schemes used in GA implementations to solve multimodal optimization problems. See [4] for more details. When $\mathcal{R} \geq \Pi$, $\rho(\xi)$ is added to $\Delta_+(\xi)$; otherwise, it is added to $\Delta_-(\xi)$.

Under this regime, rules that are consistently associated with only one type of outcome will quickly achieve a large net accumulation of resources in one of the reservoirs since all of their resources are stored in one place. Conversely, rules associated with both outcomes will distribute their resources over both reservoirs, taking longer to attain any large net accumulation. This is significant because the frequency of reproduction is tied to the net accumulation of resources in the one of the reservoirs.

2.3 Rule Discovery

After the rule reservoirs have been updated, any classifier in $\mathbf{M}$ having a sufficient net excess of resources in its reservoirs becomes eligible to reproduce. An excess of resources is indicated by

$$|\Delta_+(\xi) - \Delta_-(\xi)| > \tau$$

for some threshold τ.

If there is more than one classifier in $\mathbf{M}$ eligible to reproduce on a given cycle, all eligible classifiers are designated as parents and allowed to produce one copy of themselves. Parents then have the reservoir containing the excess decremented by τ, which can be viewed as the cost of generating an offspring. The reproduced copies are modified by mutation and crossover, and the resulting offspring are inserted into the population. Classifiers are stochastically selected for deletion based on $\alpha(\xi)$, so that general classifiers are more likely to be chosen. This is the simplest kind of deletion technique used in Echo-like systems. Future research will investigate the potential advantages of charging each rule a "maintenance cost" every time it is active, changing the resource flow to allow parents to share resources with their offspring, and deleting rules with empty (or nearly empty) reservoirs.

Note that rules consistently associated with above average (or below average) outcomes will consistently enjoy a reproductive advantage over their competitors. In combination with a deletion technique biased against general classifiers, this exerts considerable selective pressure against overly general rules.

2.4 Initial Tests

Figure 1 and Figure 2 show the performance of this revised classifier system on the 11-bit and 20-bit multiplexor problems. Results are averaged over 10 runs.

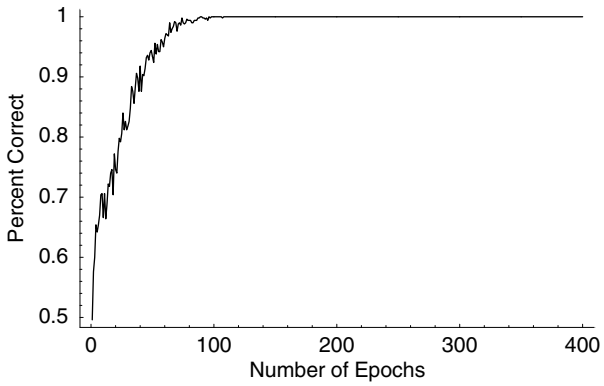

Figure 1: Performance on 11-bit multiplexor

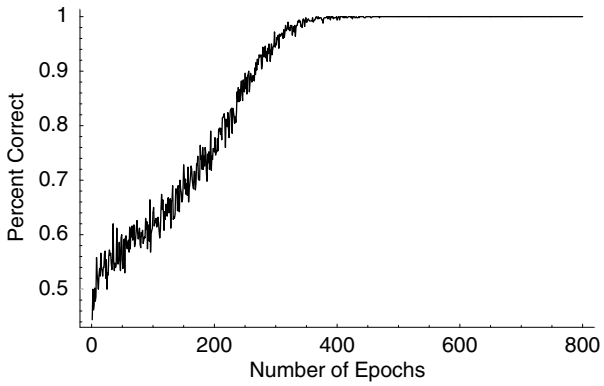

Figure 2: Performance on 20-bit multiplexor

Performance was measured by the proportion of correct decisions over a learning epoch consisting of 50 randomly generated input strings. Each 11-bit experiment was run for 400 epochs (20,000 input strings) and each 20-bit experiment was run for 800 epochs (40,000 input strings). The reward scheme pays $+1$ for correct responses and -1 for incorrect responses. The action-selection regime was based on the one used by Wilson [11]. This regime makes a random (probability 0.5) choice between "exploit" mode — where the system selects the best action as indicated by the action selection array — and "explore" mode where the system selects an action at random. During explore trials, a correct decision is recorded whenever the system would have made the correct "exploit" decision. The system parameters[1] used were: $\mathcal{N}_m = 16$, $\mathbf{R} = 2000$, $\tau = 500$, initial reservoir levels of 0 for new offspring, a mutation rate of $1/(3\ell)$, and a crossover rate of 1.0. The 11-bit experiments used $\mathcal{N} = 400$ while the 20-bit experiments used$\mathcal{N} = 800$.

Note that the 11-bit problem was solved after about 100 epochs (5,000 inputs) and the 20-bit problem was solved after about 500 epochs (25,000 inputs). This is roughly half the time reported previously for the XCS system on these problems [12]. Though it is difficult to draw any definitive conclusions on the basis of these results, it is clear that in these problems the endogenous fitness scheme does an effective job of discovering accurate rules. Work in progress is studying how this approach scales to larger multiplexor problems.

3 Accommodating Delayed Rewards

One of the important research issues not addressed by previous work on endogenous fitness in classifier systems is how to solve multi-step reinforcement learning problems involving delayed rewards. This section briefly describes an initial approach to extending the endogenous fitness scheme to handle such problems.

Classifier systems traditionally solve problems involving delayed rewards by using the bucket brigade algorithm [7] or some other algorithm from the reinforcement learning literature [9]. These algorithms all compute and manipulate explicit estimates of the reward expected when a specific action is taken in a given state. Since the endogenous fitness scheme only computes explicit estimates of the average reward expected in a state, the most natural starting point for our investigation is to consider average-payoff reinforcement learning algorithms [10].

[1] Classifier input conditions were initialized so that each possible symbol in $\{1, 0, \#\}$ was equally likely to occur.

A typical updating scheme for average-payoff reinforcement learning is given by:

$$Q_{t+1}(x_t, a_t) = (1 - \beta(x_t, a_t))Q_t(x_t, a_t) + \beta(x_t, a_t)[\mathcal{R}(x_t, a_t) - r_t + \max_{a' \in \mathbf{A}_{t+1}} Q_t(x_t, a')]$$

where x_t is the state, a_t is the action taken, $Q_t(x_t, a_t)$ is the payoff expected when taking action a_t in state x_t, and r_t is the sample average of the payoffs received for greedy actions. Note that a discounting factor is not needed to assure that the updated values remain bounded, since anchoring the computation to r_t accomplishes that.

In order to use this approach in a classifier system, we must identify something that plays the role of $Q_t(x_t, a_t)$. The endogenous fitness scheme used here only maintains explicit reward estimates associated with the match set $\mathbf{M}$, so there is no explicit information available about the payoff of an arbitrary state–action pair. However, since the resource flow experienced by a classifier is correlated with the size of the reward expected when that classifier belongs to $\mathbf{A}$, it is reasonable to consider modifying the resource flow as an alternative to updating an explicit parameter. Moreover, the parameter $\hat{\delta}_+$ provides explicit payoff information about one very important state–action pair: the one associated with the best action in $\mathbf{M}$. Consequently, the following heuristic counterpart to the average-reward reinforcement learning update is used in the endogenous fitness computation: at time t, the value $\hat{\delta}_+$ is passed back to the classifiers in $\mathbf{M}_{t-1}$ (i.e., it is added to whatever external reward was received by that match set). The computations in $\mathbf{M}_{t-1}$ then proceed as usual using the augmented reward in place of the external one.

Grefenstette's state space search problem [5] was used to test of how well this average-payoff version of the classifier system can discover action sequences leading to external reward. The state space contains 288 states arranged in a 9×32 rectangular grid. The first row contains the 32 initial states where all searches begin. Three transitions are possible from any one state to some neighboring state. If we identify each state using a row index i, $0 \leq i < 9$, and a column index j, $0 \leq j < 32$, then the states accessible from state (i, j) are the states

$$(i+1, j-1 \bmod 32) \quad (i+1, j) \quad (i+1, j+1 \bmod 32)$$

The last row in the grid contains the 32 final states, each of which is associated with a fixed reward. Rewards range from 0 to 1000 and are distributed as shown in 1.

Reward	Column Index of Final State
0	0,1,14,15,16,17,30,31
50	2,13,18,29
75	3,12,19,28
125	4,11,20,27
250	5,10,21,26
500	6,9,22,25
1000	7,8,23,24

Table 1: Distribution of rewards in the state space problem.

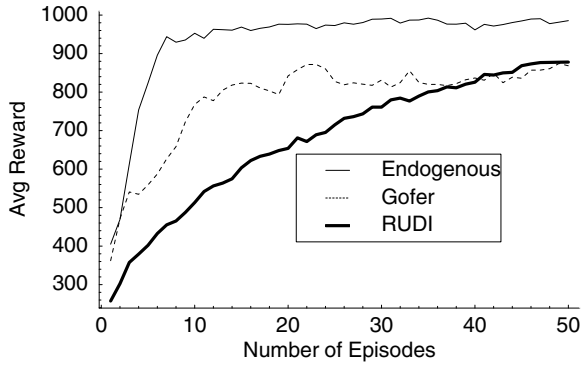

Figure 3: Performance on state space problem (compared to Gofer and RUDI)

The challenge in this problem is to learn a sequence of state transitions from each initial state that maximizes the reward obtained at the end of the sequence. It is a difficult learning problem because, from some initial states, the early moves determine whether or not it is even possible to achieve the maximum reward. Effective credit assignment is therefore a pivotal issue. Another difficulty is that there is a "hamming cliff" in the binary representation of the final states associated with the optimum reward (between columns 7 and 8, and columns 23 and 24). This complicates the categorization task faced by the genetic algorithm.

The revised classifier system was tested on this problem in an experiment involving 50 learning episodes, each consisting of 1000 traversals of the state space. The system parameters that differed from the multiplexor experiments were $\mathcal{N} = 2000$, $\mathcal{N}_m = 24$, $\mathbf{R} = 500$, and $\tau = 2500$. The action-selection regime differed slightly from the one used for the multiplexor problems. For each traversal of the state space, the system first makes a random choice between an "exploit" traversal in which the best action is taken on every step, or an "explore" traversal in which action selection is controlled by the mulitplexor action-selection

regime. The results are summarized in Figure 3, which compares the performance with previous results on this problem reported for Grefenstette's [5] system RUDI and Booker's [3] system GOFER. The revised classifier system quickly achieves good performance on this task and steadily improves toward optimum (1000 reward level) performance, clearly outperforming both RUDI and GOFER.

4 Conclusions

While these results are preliminary, they do show that an endogenous fitness scheme is compatible with reinforcement learning algorithms for problems involving delayed rewards. This makes endogenous fitness a more suitable alternative for implementing classifier systems to solve interesting problems. Moreover, the performance of the endogenous fitness approach is comparable to that obtained by systems like XCS and shows the potential to do even better. Current research efforts are conducting more experiments with this enhanced version of the endogenous fitness scheme in order to better assess its strengths and weaknesses. It is clear that this approach is promising enough to warrant further investigation.

† Acknowledgments

This research was funded by the MITRE Sponsored Research (MSR) program. That support is gratefully acknowledged.

References

[1] Vic Barnett and Toby Lewis. *Outliers in Statistical Data*, Third edition. John Wiley and Sons, Chichester UK, 1994.

[2] Lashon B. Booker. Classifier systems that learn internal world models. *Machine Learning*, 3:161–192, 1988.

[3] Lashon B. Booker. Triggered rule discovery in classifier systems. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA89)*, pages 265–274, Fairfax, VA, 1989. Morgan Kaufmann.

[4] Lashon B. Booker. Do We Really Need to Estimate Rule Utilities in Classifier Systems? In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems: From Foundations to Applications*, volume 1813 of *LNAI*, pages 125–142, Berlin, 2000. Springer-Verlag.

[5] John J. Grefenstette. Credit assignment in rule discovery systems based on genetic algorithms. *Machine Learning*, 3:225–245, 1988.

[6] John H. Holland. Echoing emergence: Objectives, rough definitions, and speculations for Echo-class models. In G. Cowan, D. Pines, and D. Melzner, editors, *Complexity: Metaphors, Models, and Reality*, volume XIX of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 309–342. Addison-Wesley, Reading, MA, 1994.

[7] John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, Cambridge, 1986.

[8] Boris Iglewicz and David. C. Hoaglin. *How to Detect and Handle Outliers*, volume 16 of *American Society for Quality Control Basic References in Quality Control: Statistical Techniques.* ASQC Quality Press, Milwaukee WI, 1993.

[9] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[10] Satinder P. Singh. Reinforcement learning algorithms for average-payoff Markovian decision processes. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 700–706, Seattle, WA, 1994. The AAAI Press.

[11] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[12] Stewart W. Wilson. State of XCS Classifier System Research. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems: From Foundations to Applications*, volume 1813 of *LNAI*, pages 63–81, Berlin, 2000. Springer-Verlag.

How XCS Evolves Accurate Classifiers

Martin V. Butz
Department of Cognitive Psychology
University of Würzburg
Würzburg, 97070, Germany
butz@psychologie.uni-wuerzburg.de
+49 931 312176

Tim Kovacs
School of Computer Science
The University of Birmingham
Birmingham B15 2TT, United Kingdom
T.Kovacs@cs.bham.ac.uk
+44 121 414 4773

Pier Luca Lanzi
Dip. di Elettronica e Informazione
Politecnico di Milano
Milano 20133, Italy
pierluca.lanzi@polimi.it
+39 2 2399 3472

Stewart W. Wilson
University of Illinois at Urbana-Champaign
Prediction Dynamics
Concord, MA 01742, USA
wilson@prediction-dynamics.com
+1 978 369 9232

Abstract

Due to the accuracy based fitness approach, the ultimate goal for XCS is the evolution of a compact, complete, and accurate payoff mapping of an environment. This paper investigates what causes the XCS classifier system to evolve accurate classifiers. The investigation leads to two challenges for XCS, the *covering challenge* and the *schema challenge*. Both challenges are revealed theoretically and experimentally. Furthermore, the paper provides suggestions for overcoming the challenges as well as investigates environmental properties that can help XCS to overcome the challenges autonomously. Along those lines, a deeper insight into how to set the initial parameter values in XCS is provided.

1 INTRODUCTION

After more than twenty years have passed since the first learning classifier system (LCS) approaches (Holland & Reitman, 1978), recently, LCSs appear to reach competence. The XCS classifier system (Wilson, 1995) solved the former main shortcoming of LCSs, that is the problem of strong over-generals (Kovacs, 2001), by its accuracy based fitness approach. Previous LCSs evolved strong rules in terms of rules that encounter high rewards from environments. On the other hand, XCS evolves accurate rules, i.e., rules which accurately predict the payoff encountered after the execution of an action.

Despite this insight, til now it has not been clarified how the genetic algorithm (GA) in XCS can benefit from this approach. Essentially, it is not clear how and when the accuracy based fitness approach pushes the population of classifiers towards accurate classifiers. The aim of this paper is to investigate and clarify the evolutionary pressure in XCS towards accurate classifiers. Along those lines the paper exposes two problem boundaries or challenges that can more or less severely decrease the accuracy pressure. Alternatives how to circumvent the problem are provided. Moreover, environmental properties are investigated that help XCS to evolve accurate classifiers faster. Finally, the utility and best initial setting of several parameters is clarified.

The paper is structured as follows. First, we provide a short overview over the XCS classifier system mentioning all parameters and equations important for the remainder of the paper. Section 3 discusses the accuracy pressure in XCS theoretically, emphasizing two challenges and possible solutions. Next, section 4 validates the proposed pressures and solutions. Moreover, it provides further insight in the best initial parameter setting. Finally, we summarize and conclude the paper.

2 XCS OVERVIEW

The XCS classifier system was developed by Wilson (1995). Although we assume a basic familiarity with

the system, this section provides a general overview of XCS displaying the accuracy related methods in detail. For further information the interested reader should refer to Wilson (1995), and the algorithmic description of XCS (Butz & Wilson, 2001).

As in all LCSs and reinforcement learning methods, XCS acts as a learning agent that perceives inputs describing the current environmental state, responds with actions, and receives reward (possibly from a separated reinforcement program) as an indication of the value of its action. The reward received is defined by the *reward function*, which maps state/action pairs to real numbers, and it is part of the problem definition (Sutton & Barto, 1998). For the investigation purposes in this paper we only use *single-step* tasks in which the agent's actions do not influence the successive states. The goal of the agent is to maximize the reward it receives.

When XCS receives an input it forms the *match set* [M] of rules whose conditions match the environmental input. XCS requires that at least θ_{mna} actions are present in a match set (Butz & Wilson, 2001). If this is not the case, covering classifiers will be created with a matching condition. Each attribute in the condition of such a covering classifier is a #-symbol (a so called don't care symbol that matches any input) with a probability of $P_{\#}$ and the corresponding perceived symbol otherwise. Next, XCS selects an action from among those advocated by the rules in [M]. The subset of [M] which advocates the selected action is called the *action set* [A].

In each cycle, XCS updates the rules in [A] based on the reward received. Rules not in [A] are not updated. Moreover, dependent on the threshold θ_{ga} and the average time in $[A]$ since the last GA application, a reproductive event is triggered, in which a GA is called upon to modify the population of rules. Since the GA in XCS only reproduces classifiers currently in [A] it realizes an implicit niching. The GA chooses two classifiers for reproduction proportionally to the fitnesses F of the classifiers in $[A]$. The selected classifiers are reproduced, crossed, mutated, and inserted in the population. The parents stay in the population competing with their offspring. Moreover, subsumption deletion acts in [A] deleting more specific classifiers if an accurate, experienced, and more general classifier exists.

If the number of classifiers in a population exceeds the threshold N, excess classifiers are deleted. Classifiers for deletion are selected in [P] proportionally to their action set size estimate *as*. If sufficiently experienced and with a significantly low fitness F, the probability of deletion is increased further.

The rule fitness calculation in XCS differs from traditional approaches. In traditional strength-based systems (e.g., Goldberg, 1989; Wilson, 1994), the fitness of a rule is called its *strength*. This value is used in both action selection and reproduction. In contrast, the accuracy-based XCS maintains separate estimates of rule utility for action selection and reproduction.

In single-step tasks an LCS typically uses an update like the following delta rule to update rule strength which is called the reward prediction p in XCS:

$$p \leftarrow p + \beta(R - p) \tag{1}$$

where $0 < \beta \leq 1$ is a constant controlling the learning rate and R is the reward from the environment. From the reward prediction p XCS updates a number of parameters for adjusting its fitness F:

$$\epsilon \leftarrow \epsilon + \beta(|R - p| - \epsilon) \tag{2}$$

$$\kappa = \begin{cases} 1 & \text{if } \epsilon < \epsilon_0 \\ \alpha(\epsilon/\epsilon_0)^{-\nu} & \text{otherwise} \end{cases} \tag{3}$$

$$\kappa' \leftarrow \frac{\kappa}{\sum_{x \in [A]} \kappa_x} \tag{4}$$

$$F \leftarrow F + \beta(\kappa' - F) \tag{5}$$

The parameter ϵ_0 ($\epsilon_0 > 0$) controls the tolerance for prediction error ϵ; α ($0 < \alpha < 1$) and ν ($\nu > 0$) are constants controlling the rate of decline in accuracy κ when ϵ_0 is exceeded. The updates treat the strength of a rule as a prediction of the reward to be received, and maintain an estimate of the error ϵ of its reward prediction. An accuracy score κ is calculated based on the error as follows. If error is below some threshold ϵ_0 the rule is assumed to be accurate (has an accuracy of 1), otherwise its accuracy drops off quickly. The accuracy values in the action set [A] are then converted to relative accuracies κ', and finally each rule's fitness F is updated towards its relative accuracy. Figure 1 visualizes equation (3). Observable is the idea behind the accuracy calculation: ϵ_0 is the threshold to what extent errors are accepted; α causes a strong distinction between accurate and not quite accurate classifiers; the steepness of the successive slope is influenced by ν as well as ϵ_0.

To summarize, in XCS fitness behaves inversely to the reward prediction error, with errors below ϵ_0 being ignored entirely.

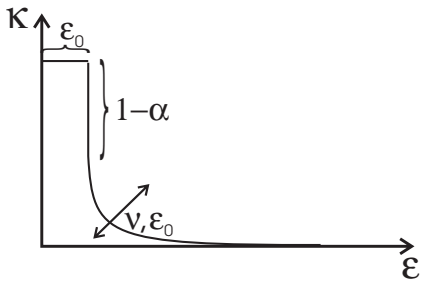

Figure 1: The calculation of the accuracy kappa is crucial for the fitness approach in XCS.

3 FINDING ACCURATE CLASSIFIERS

Although the last section gave an overview of the functioning of XCS, it is not clear how the GA method evolves accurate classifiers. The accuracy based fitness is only a prerequisite for a successful evolution. This section explains first how the pressure towards accurate, maximally general classifiers (i.e. classifiers which are accurate and in the mean time as general as possible) is realized in XCS. Next, it investigates problem boundaries that hinder parts of the pressures and consequently hinder XCS from evolving accurate classifiers. Finally, solutions to those boundaries are proposed.

3.1 THE ACCURACY PRESSURE

The accuracy pressure in XCS is realized by several methods based on the principle of the survival of the fittest and die out of the weak. However, minor modifications and interpretation of these principles give XCS the real power. In the following, we first describe the methods separately and next discuss their interaction.

The reproduction method in the GA realizes the survival of the fittest principle. Reproduced are classifiers with a high fitness which implies a higher accuracy than other classifiers in the same environmental niche. Thus, accurate rather than over-general classifiers are reproduced. (We use the term *environmental niche* referring to one necessary solution of a problem described by a *schema* of lowest possible order (see e.g. Goldberg, 1989) together with its incidental action. The order o of a schema denotes the number of specific attributes in the schema.)

The deletion method, separated from the reproduction process, emphasizes the deletion of classifiers in large niches. Moreover, applying the deletion method in Kovacs (1999), deletion further emphasizes the deletion of inaccurate classifiers.

Together, reproduction and deletion realize a pressure towards the evolution of accurate classifiers in each environmental niche. Moreover, due to the application of the reproduction in the action set and the deletion in the population, the combination also realizes an intrinsic pressure towards more general classifiers as proposed by Wilson's Generality Hypothesis (Wilson, 1995), which was further investigated and enhanced to an Optimality Hypothesis by Kovacs (1997).

This combination of accuracy and generality pressure, however, requires several conditions in order to be applicable. The prerequisites are discovered in the remainder of this section.

3.2 COVERING CHALLENGE

The first rather straightforward challenge is to set the parameters in such a way that the GA actually takes place in XCS. One case in which this prerequisite is not met is expressed by the *covering challenge*. Normally, covering only occurs briefly at the beginning of a run. However, if covering continues indefinitely because inputs continue not to be covered by classifiers, the GA cannot take place in XCS and the accuracy pressure does not apply.

As described in the XCS overview above, dependent on the parameter θ_{mna}, one or more covering classifiers will be created if an input is not sufficiently covered. However, if the population is already filled up with classifiers, other classifiers are deleted to make space for the covering classifiers. In the beginning of a run, with a population of classifiers that have a very low experience, the fitness F as well as the action set size estimate *as* of these classifiers is basically meaningless. Consequently, the deletion method chooses classifiers for deletion at random. Dependent on the specificity of classifiers generated by covering (determined by the parameter $P_\#$) the population has on average a certain specificity in the beginning. If this specificity is too high, it can happen that the population is eventually filled up with over-specific classifiers and the "covering-random deletion" cycle continues forever.

More formally, the probability that an input is covered by at least one classifier in a randomly generated population is:

$$P(\text{cover}) = 1 - (1 - (\frac{1 + P_\#}{2})^l)^N \qquad (6)$$

where $P_\#$ is the probability of generating a don't care symbol when creating the condition of a classifier, l is the length of the input string, and N is the size of the population. The formula is correct for binary coding where the population was initially filled up with

randomly generated classifiers. Moreover, the formula also applies in the case where $P_\#$ is too small, so that covering will continue until the population is filled up with different classifiers, and all expressible inputs are equally probable encountered without any dependence on previous states or actions. Note that, in multi-step problems $P(\text{cover})$ varies further since not all possible codings are perceivable and states closer to a goal state are usually encountered more often.

The covering challenge by itself can easily be circumvented by setting the parameter $P_\#$ high enough. Considering formula (6) it is possible to calculate a curve dependent on $P_\#$. As long as $P(\text{cover})$ is sufficiently larger than zero, the GA eventually takes place and the covering challenge is solved. Moreover, it should be possible to enhance XCS to detect the challenge itself and consequently adapt the $P_\#$ parameter autonomously. However, we did not experiment with such an enhancement so far.

3.3 SCHEMA CHALLENGE

Once the covering challenge is solved and the GA takes place, essential for a successful evolutionary process is that the accuracy pressure applies. The *schema challenge* addresses the problem that the accuracy of classifiers possibly does not lead to the accurate, maximally general point. Considering again the specificity of a classifier, this point can be reached from two sides, the over-specific side and the over-general side.

3.3.1 THE OVER-SPECIFIC SIDE

Evolving an accurate, maximally general classifier from the over-specific side is a process that is based on Wilson's *generalization hypothesis* as mentioned in section 3.1. Once an accurate classifier was generated (which can happen by chance in covering or by evolution in the GA) this classifier will eventually have a higher fitness than the majority of the inaccurate classifiers and consequently reproduce more often. To what extent the reproduction rate of accurate classifiers differs from inaccurate ones depends on the parameters α and ν in the determination of the accuracy κ as visualized in figure 1. Setting α low enough and ν high enough ensures that the probability of loosing all accurate classifiers in a specific environmental niche is small. Thus, the intrinsic pressure expressed in the generalization hypothesis ensures a continuous pressure towards generality while the accuracy based fitness prevents from over-generalization. The subsumption deletion method increases the pressure towards accurate, maximally general classifiers from the over-specific side further.

3.3.2 THE OVER-GENERAL SIDE

While the pressure towards generality was already intensively investigated, the evolution from the over-general side appears to be much more awkward. How can an over-general classifier be fitter, when closer to being accurate? The distance is hereby the number of attributes that need to be changed in order to generate an accurate classifier out of the current classifier. The remainder of this section exposes necessary requirements in the environment. Moreover, if those requirements are not met, it exhibits how to circumvent the problem at least in small problem spaces.

Essential for a higher fitness of classifiers which are closer to accuracy is the applicability of the slope in the determination of κ as visualized in figure 1. Thus, it is necessary that the prediction error ϵ of classifiers which are closer to accuracy is lower than the ϵ of classifiers with a higher distance. Two environmental properties can provide such 'hints' from the over-general side. (1) The reinforcement program can provide intrinsic information in its reward function. (2) A bias in the consistency of making a correct/wrong classification when over-general can result in further benefit. Both possibilities are discussed in somewhat more detail below.

Layered Payoff. The first property assumes a bias in the reward function of the reinforcement program where a bias refers to the reward function returning different payoff in different states. The resulting layered payoff landscapes can provide 'hints' towards accuracy. Wilson (1995) used such a landscape for the multiplexer problem and Kovacs (2001) used them to develop a theory of strong over-generals calling a function that creates such landscapes a biased reward function. Moreover, layered payoff landscapes are always present in multi-step problems. In order to be helpful for a problem the difference in payoff between a correct and wrong classification must be on average smaller when closer to accuracy. If this property is present in any payoff landscape, XCS is able to exploit the property and consequently discovers the accurate classifiers faster as experimentally validated in section 4.

Biased Generality. While the layered payoff benefit requires an explicit bias in the reward function of a problem, the *biased generality benefit* can also be intrinsically present in a problem even when the reward function is not biased. The idea behind biased generality is that accuracy is greater when a rule is more consistently correct or more consistently wrong. Once a classifier is only correct or wrong it is most accurate. The property can be approached mathematically when assuming a simple $R/0$ (i.e. a two level) payoff landscape, R is provided if a prediction was cor-

rect and zero if it was wrong. Let's denote $P_c(cl)$ as the probability that classifier cl predicts the correct outcome. Due to the assumed payoff landscape and the assumption of a uniformly randomly encountering of both cases, the reward prediction $cl.p$ of classifier cl eventually oscillates around $P_c(cl) * R$ where the amount of oscillation can be influenced by β. Neglecting the oscillation and consequently setting $cl.p$ equal to $P_c cl * R$ the following derivation is possible:

$$cl.\epsilon = (R - cl.p) * P_c(cl) + cl.p * (1 - P_c(cl)) = \\ = 2R(P_c(cl) - P_c(cl)^2) \quad (7)$$

The formula sums the two cases of executing a correct or wrong action with the respective probabilities. The result is a parabolic function for the error ϵ that reaches its maximum of 0.5 when $P_c(cl)$ equals 0.5 and is 0 for $P_c(cl) = 0$ and $P_c(cl) = 1$. This shows that if the consistency of a correct/wrong prediction increases, the accuracy and consequently the fitness of a classifier increases.

3.3.3 CIRCUMVENT THE CHALLENGE

However, when neither of the two above properties are given in a problem (i.e. the problem does not provide any hints towards accuracy from the over-general side), it can be very hard for XCS to evolve accurate classifiers out of an over-general population of classifiers. One possibility to circumvent this problem is to set $P_\#$ low enough, so that accurate classifiers for most environmental niches are present due to covering.

Hereby, it is helpful to calculate the probability that a certain environmental niche is represented by at least one classifier. We can think of an environmental niche as a schema (see e.g. Goldberg, 1989) of order o combined with an action. An environmental niche is represented by a classifier if at least all attributes that are specified in the schema are equal in the classifier and the classifier has the same action. The probability of the existence of a representative of a specific schema is determined by the formula

$$P(\text{representative}) = 1 - (1 - \frac{1}{n}(\frac{1 - P_\#}{2})^o)^N \quad (8)$$

where n denotes the number of possible actions. As in equation (6) the equation again assumes a specificity in the population that is identical to $P_\#$. Thus, the schema challenge can be initially circumvented by setting $P_\#$ low enough so that $P(\text{representative})$ is significantly larger than zero.

While the covering challenge requires to set $P_\#$ high enough, the schema challenge—as long as no hints are given from the environment—actually requires $P_\#$ to be low enough. The next section validates all above discussed challenges and properties. Moreover, it shows that the two challenges can interfere consequently hindering XCS from evolving accurate classifiers.

4 EXPERIMENTAL VALIDATION

After many claims have been made in the last section, this section is dedicated to validate all the claims experimentally. In order to show the problem boundaries induced by the two challenges, we first investigate performance in the 20 multiplexer varying $P_\#$. Next, we show that despite the increased boundaries, XCS is able to solve the 37 multiplexer. Moreover, we show the benefit of a biased reward function in the 37 multiplexer and reveal the use of the parameters α and ϵ_0 in the κ function. With the biased reward function we are now even able to solve the 70 multiplexer. Finally, we modify the multiplexer function in order to show the benefit of biased generality.

If not stated differently, all experiments herein are averaged over twenty runs. Performance is measure by altering one pure exploration step with one pure exploitation step in which the percentage of correct classifications is recorded. The different methods in XCS go along with the recently published algorithmic description (Butz & Wilson, 2001). Essentially, we use the niche mutation type, the deletion method as investigated by Kovacs (1999), and GA- as well as action set subsumption. The parameters were set as follows: $N = 2000$, $\beta = 0.2$, $\alpha = 0.1$, $\epsilon_0 = 10$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, $p_I = 10$, $\epsilon_I = 0$, $F_I = 0.01$, $p_{explr} = 1$, and $\theta_{mna} = 2$. Note that γ is irrelevant since we only investigate single-step problems. Parameter $P_\#$ is set in each experiment separately.

4.1 THE TWO CHALLENGES

Before we present the challenges in the problem, we need to explain the multiplexer problem itself. The multiplexer function is defined for binary strings of length $k + 2^k$. The output of the function is determined by the bit referred to by the first k bits. In the six multiplexer for example, $f_{6-MP}(100010) = 1$ or $f_{6-MP}(000111) = 0$. Any multiplexer function can also be expressed in disjunctive normal form. The six multiplexer can be expressed as:

$$f_{6-MP}(x_1, ..., x_6) = \\ x_1 x_2 x_6 \vee x_1 \neg x_2 x_5 \vee \neg x_1 x_2 x_4 \vee \neg x_1 \neg x_2 x_3 \quad (9)$$

The symmetry of the multiplexer problem results in most over-general cases in no generality bias at all (i.e. the probability for an over-general classifier of being

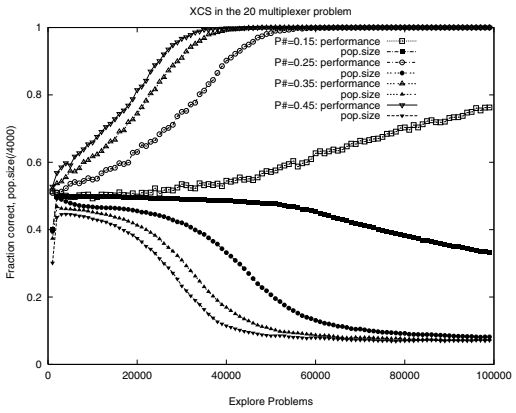

Figure 2: In the 20 multiplexer problem the covering challenge is easily observable.

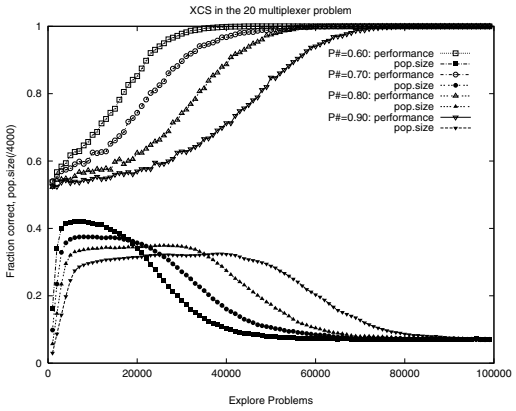

Figure 3: Despite the observable schema challenge XCS is always able to solve the problem.

correct is very close to 50%). Providing a 1000/0 payoff the payoff is not biased and no benefit can be drawn in this respect, either. Thus, the cover challenge from the over-specific side as well as the schema challenge from the over-general side should be observable.

Figure 2 exhibits the covering challenge. If the don't care probability $P_\#$ is set too low, the population is filled up with classifiers and has difficulties to start with the GA application. Increasing $P_\#$ results in a faster learning curve as well as a faster convergence in the population. Due to the generalization pressure, the population size decreases once the GA starts being applied and XCS is able to evolve a complete and accurate representation of the problem. The relation with formula (6) is observable in figure 4. At a don't care probability of 0.15 the curve is very close to zero consequently causing the covering challenge. With increasing $P_\#$ the curve increases and the covering challenge diminishes.

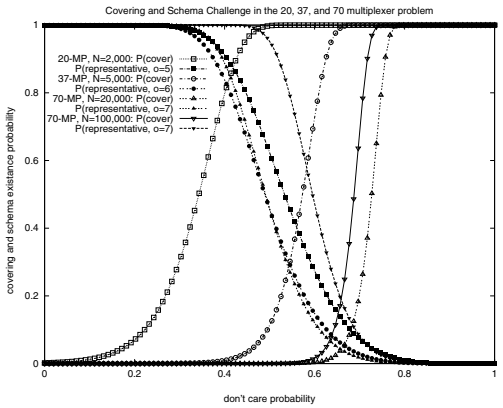

Figure 4: The two challenges visualized for the 20, 37, and 70 multiplexer problem.

Similarly, the schema challenge is observable in figure 3 although not quite as strong as the covering challenge. When $P_\#$ comes closer to one, the population is initially over-general and it is consequently more difficult to evolve accurate classifiers. Note that the schema challenge appears to make life harder for XCS until $P_\# = 0.6$ which shows that the drawback is not simply due to a further distance from the accurate, maximally general classifiers which have a generality of 0.75 in the 20-multiplexer. Figure 4 shows a close match between schema challenge curve and the results in the 20 multiplexer.

4.2 LAYERED PAYOFF BENEFIT

As visualized in figure 4 the window that allows a solution to the problem severely diminishes in the 37multiplexer. However, XCS is still able to find a solution as visualized in figure 5.

Moreover, figure 5 shows a strongly increased accuracy pressure once the reward function is changed resulting in a layered payoff landscape. The biased reward function uses the formula (value of the k position bits+return value) $* 100 +$ (correctness) $* 300$ as used in Wilson (1995). The consequence of the formula is that any specification of one of the relevant position bits, results in a decrease of the number of different rewards and consequently to a decrease in the prediction error. This decrease results in an increase of selecting classifiers which are closer to accuracy. The strong benefit of this increase is observable in figure 5. While XCS in the multiplexer with 1000/0 payoff reaches a 100% knowledge after about $500,000$ problems, XCS in the 37 multiplexer with the biased reward function actually reaches a 100% knowledge after about $100,000$ problems. Note also the differences when al-

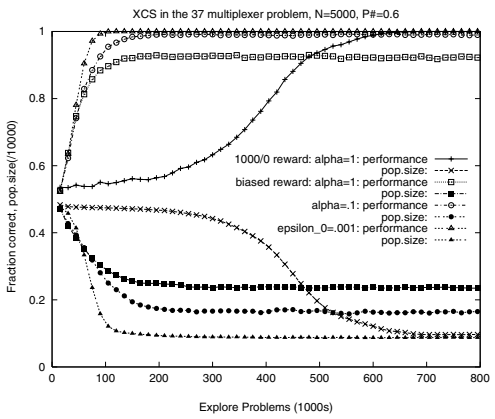

Figure 5: The 37 multiplexer problem is solvable for an appropriate parameter setting. When introducing a biased reward function, epsilon needs to be lowered.

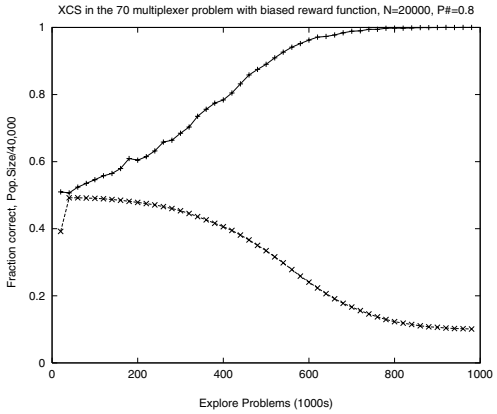

Figure 6: XCS is able to solve the 70 multiplexer problem if a biased reward function is provided.

tering the parameters α and ϵ_0 in the calculation of κ. The stronger difference when decreasing α and ϵ_0 enables XCS to distinguish the different reward levels stronger. Thus, it evolves an accurate performance faster and stays more reliably on a 100% performance level.

The layered payoff benefit even enables XCS to solve the 70multiplexer problem as shown in figure 6. So far, we didn't succeed to solve the 70multiplexer problem without layered payoff. Although the two challenges decrease the space for possible solutions as shown in figure 4 it seems theoretically possible for XCS to find a solution if a sufficiently large population is provided. However, since the first environmental niches can only be found with low chance, the solution of the 70 multiplexer without layered payoff appears to be hard.

4.3 BIASED GENERALITY BENEFIT

After we revealed the possible benefit due to a biased reward function, what remains is to show the benefit due to a change in the correct prediction when getting closer to accuracy. In order to uncover this benefit, we decided to modify the multiplexer function resulting in what we call a *xy-biased multiplexer function*. This function forms a hierarchy of depth two. The first x-position bits refer to one of the 2^x biased multiplexers located in the remaining $l - x$ bits. A biased multiplexer is defined for all $l = y + 2^y - 1$ since the biased multiplexer is always one if all y position bits are one or always zero if all its y position bits are zero dependent on if the value of the first x position bits is bigger or smaller than $(2^x - 1)/2$, respectively. The result is a biased consistency on several generality levels in over-general classifiers.

Figure 7 shows that XCS can benefit from such a bias. Shown are runs with $x = 1$ and $y = 3$, $x = 2$ and $y = 2$, and $x = 3$ and $y = 1$ which we refer to as $1,3$, $2,2$ and $3,1$ respectively. The don't care probability is set to 0.95 to assure that there are no accurate classifiers in the beginning of a run. Having a look at the previously proposed difficulty measure in Kovacs and Kerber (2001) (i.e. the size of a population that covers all environmental niches accurately, is non-overlapping, and minimal, denoted by $|[O]|$) we can observe that $|[O]|(3,1) = 48$, $|[O]|(2,2) = 56$ and $|[O]|(1,3) = 60$. The figure confirms that this is indeed a crucial measure. Moreover, we can observe that the actual problem length l seems not to have a broad impact on performance, since $l(3,1) = 19$, $l(2,2) = 22$, and $l(1,3) = 21$. However, why the performance in all three cases is much better than the performance in the 20 multiplexer where $|[O]| = 64$ and $l = 20$ is not explainable by either measure. Also the fact that the $2,2$ problem appears to be similarly difficult to the $3,1$ problem is not explainable. Finally, the plateau in the $3,1$ curve is not explainable, either. All three peculiarities however show that XCS benefits from the percentage bias. Despite the low specificity in the beginning of all runs, XCS is able to evolve the necessary specializations fast. Moreover, despite the higher $|[O]|$ measure in the $2,2$ run, XCS is able to further benefit from the percentage bias in this setting and consequently reaches a 100% knowledge as fast as in the $3,1$ run. Finally, the plateau in the $3,1$ run reveals that XCS first discovers the necessary specificity of the first bit which results, if specified in a 0.75% correctness of a classifier. However, due to the minor bias in the remaining four to be specified bits, it takes longer to proceed to a 100% knowledge. The $2,2$ case has a lower bias in the specification of the first bit, but

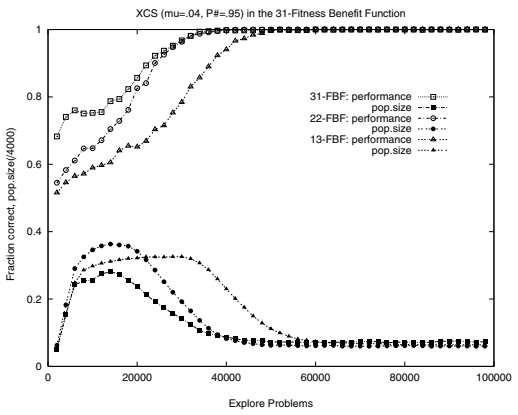

Figure 7: The increasing consistency in classification when getting closer to being accurate helps XCS in evolving a complete and accurate representation.

a stronger one, in the remaining specifications. Thus, in this case it takes longer to reliably evolve the first specifications but faster to specify the rest.

5 SUMMARY AND CONCLUSION

This paper investigated how XCS evolves accurate classifiers. We showed that the accuracy based fitness approach in XCS together with the generality pressure causes an evolutionary pressure towards accurate, maximally general classifiers. However, several problems were encountered that can prevent XCS from finding a complete and accurate solution to a given problem. The *covering challenge* needs to be met in order to prevent XCS from getting stuck in a continuous covering - deleting loop without any sort of evolution. The *schema challenge* faces XCS with the problem of evolving classifiers from the over-general side. Two environmental properties proved to help XCS in this endeavor. (1) *Layered payoff benefit* can be encountered if specialization of relevant bits leads to an on average smaller difference in payoff encountered. (2) *Biased generality benefit* is the result of a bias in the consistency of a correct/wrong classification in over-general classifiers. Moreover, it has been shown that the accuracy function is crucial in exploiting the benefits and evolving a complete and accurate solution to a problem.

Although most of the investigations herein appear to be rather theoretical, we hope that our approach leads to a broader understanding of XCS. Moreover, the two challenges together with the formulas provide rules of thumb how to set several initial parameters in XCS. Finally, the paper provides several insights how to include background knowledge in the system. By introducing a bias in the reward function, XCS can get hints that lead the system to a certain direction. Future research will show to what extent this characteristic is exploitable in the system.

References

Butz, M. V., & Wilson, S. W. (2001). An algorithmic description of XCS. *In Lanzi, P. L., Stolzmann, W. and Wilson, S. W. (Eds.), Advances in Learning Classifier Systems, LNAI 1996*. to appear.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning.* Reading, Massachusetts: Addison-Wesley.

Holland, J. H., & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In Waterman, D. A., & Hayes-Roth, F. (Eds.), *Pattern Directed Inference Systems* (pp. 313–329). New York: Academic Press.

Kovacs, T. (1997). XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, & Pant (Eds.), *Soft Computing in Engineering Design and Manufacturing* (pp. 59–68). Springer-Verlag, London.

Kovacs, T. (1999). Deletion schemes for classifier systems. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)* (pp. 329–336). San Francisco, CA: Morgan Kaufmann.

Kovacs, T. (2001). Towards a theory of strong overgeneral classifiers. *In Fogarty, T. C., Martin, W. and Spears, W. M. (Eds.), Proceedings of the Workshop on Foundations of Genetic Algorithms (FOGA2000)*. to appear.

Kovacs, T., & Kerber, M. (2001). What makes a problem hard for XCS? *In Lanzi, P. L., Stolzmann, W. and Wilson, S. W. (Eds.), Advances in Learning Classifier Systems, LNAI 1996*. to appear.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press.

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.

Analyzing the Evolutionary Pressures in XCS

Martin V. Butz
Department of Cognitive Psychology
University of Würzburg
Würzburg, 97070, Germany
butz@psychologie.uni-wuerzburg.de

Martin Pelikan
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
Urbana-Champaign, IL, USA
pelikan@illigal.ge.uiuc.edu

Abstract

After an increasing interest in learning classifier systems and the XCS classifier system in particular, this paper locates and analyzes the distinct evolutionary pressures in XCS. Combining several of the pressures, an equation is derived that validates the generalization hypothesis which was stated by Wilson (1995). A detailed experimental study of the equation exhibits its applicability in predicting the change in specificity in XCS as well as reveals several other specificity influences.

1 INTRODUCTION

The accuracy based fitness approach in XCS (Wilson, 1995) results in a learning classifier system (LCS) that evolves not only classifiers for best actions, but a complete *payoff map* of the problem. This means that the system evolves an internal representation that can determine the quality of each possible action in each possible state of the encountered environment. Several studies showed that the payoff map in XCS is compact, complete, and accurate.

The purpose of this paper is to clarify and analyze the evolutionary pressures in XCS. The combination of several pressures results in a formula that predicts the change in specificity in the population. This formula validates the generalization hypothesis (Wilson, 1995), which was experimentally investigated in Kovacs (1997). Providing experimental evidence, the formula proves its applicability in an over-general population as well as an accurate one.

The paper starts with an overview over XCS with all involved processes relevant for the paper. Next, the evolutionary pressures in XCS are first analyzed separately and then in interaction. Section 5 provides experimental validation of the claimed pressures, interactions, and parameter dependencies. Finally, a conclusion is provided.

2 XCS OVERVIEW

The XCS classifier system, as it is explained and used herein, incorporates the basics published by Wilson (1995) and the further enhancements in Wilson (1998) and Kovacs (1999). An algorithmic description of the used system can be found in Butz and Wilson (2001). This section gives an overview of XCS emphasizing the formulas and methods important in the remainder of the paper. For further details the interested reader should refer to the cited literature.

As all LCSs the XCS interacts with an environment. The environment provides situations or problem instances σ coded as binary strings of length L (i.e. $\sigma \in \{0,1\}^L$). Furthermore, actions $\alpha \in \alpha_1, ..., \alpha_n$ are executable in the environment. Finally, the environment provides a scalar reward $\rho \in \Re$ reflecting the correctness or quality of the last applied action.

As all LCSs, XCS consists of a population $[P]$ of classifiers which is of fixed length N. The structure of a classifier in XCS is as follows. The *condition part* C specifies where the classifier is applicable. It is coded as a string over the ternary alphabet $\{0,1,\#\}$ of length L (i.e. $C \in \{0,1,\#\}^L$). The *action/classification part* A specifies the action/classification of the classifier. It can specify any action executable in the environment ($A \in \alpha_1, ..., \alpha_n$). The *reward prediction* p estimates the payoff encountered after the execution of the specified action. The *prediction error* ϵ estimates the current error of p and is essentially used for the accuracy and resulting fitness determination. The *fitness* F is a measure of the accuracy of p with respect to all competing classifiers. The *experience exp* counts how of-

ten the parameters of the classifier were updated. The *time stamp ts* stores the time when last the classifier was in a set where a GA was applied. The *action set size* estimate *as* approximates the average size of the action sets the classifier belongs to. The *numerosity num* reflects how many micro-classifiers (usual classifiers) this *macroclassifier* represents. This notation is only important for efficiency purposes.

At the beginning of an experiment the population of XCS is usually empty. Sometimes though, the population is initialized with randomly generated classifiers. Each attribute in the condition of such classifiers is set to a #-symbol (a "don't care"-symbol) with a probability $p_\#$ and to zero or one (chosen randomly) otherwise. The action is chosen randomly among all possible actions.

A learning cycle at time step t starts with the perception of the actual problem $\sigma(t)$ and the consequent formation of the match set $[M]$. If less than θ_{mna} actions are represented in $[M]$, *covering* occurs. In covering, a matching classifier is created similar to the procedure when initializing the population. Next, an action a is selected randomly with a probability of p_{explr} and deterministic otherwise. Out of $[M]$ an action set $[A]$ is formed consisting of all classifiers that specify action a. The action is executed in the environment and a reward $\rho(t)$ is perceived. With respect to the perceived reward (and the maximal reward prediction in the successive match set in multi-step problems), the reward prediction p, the error measure ϵ, and the action set size estimate as of all classifiers are updated using the Widrow-Hoff delta rule (Widrow & Hoff, 1960).

$$p_{cl} = p_{cl} + \beta * (\rho - p_{cl}) \tag{1}$$

$$\epsilon_{cl} = \epsilon_{cl} + \beta * (|\rho - p_{cl}| - \epsilon_{cl}) \tag{2}$$

$$as_{cl} = as_{cl} + \beta * (\sum_{c \in [A]} num_c - as_{cl}) \tag{3}$$

Parameter $\beta \in (0,1)$ denotes the learning rate. If the experience of a classifier is still less than $1/\beta$, p, ϵ, and as are updated with the MAM technique ("moyenne adaptive modifiée") which sets the values to the averaged actual values encountered so far. The fitness is updated in three steps.

$$\kappa_{cl} = \begin{cases} 1 & \text{if } \epsilon_{cl} < \epsilon_0 \\ \alpha * (\epsilon_{cl}/\epsilon_0)^{-\nu} & otherwise \end{cases} \tag{4}$$

$$\kappa'_{cl} = \frac{\kappa_{cl} * num_{cl}}{\sum_{c \in [A]} \kappa_c * num_c}$$

$$F_{cl} = F_{cl} + \beta * (\kappa'_{cl} - F_{cl}) \tag{5}$$

First, the accuracy κ is calculated according to the current prediction error ϵ. Next, the relative accuracy κ' is calculated with respect to the current action set. Finally, the fitness is updated according to κ'. Note that the fitness is calculated in terms of *macroclassifiers* while the value of all other measures specifies the micro-classifier value. After all updates and the increase of the experience counter exp of each classifier, a GA may be applied.

The GA is only applied if the average time in the action set $[A]$ since the last GA application, recorded by the time stamp ts, is greater than the threshold θ_{GA}. If a GA is applied two classifiers are selected in $[A]$ for reproduction using a roulette wheel selection with respect to the fitness of the classifiers in $[A]$. Next, the classifiers are reproduced and the children undergo mutation and crossover. In mutation, each attribute in C of each classifier is changed with a probability μ. Two mutation types are investigated herein. In *niche mutation* classifiers are only mutated in such a way that they still match the current state after mutation (i.e. #-symbols are mutated to the current corresponding input value and 1s and 0s are mutated to #-symbols. In *free mutation* an attribute is mutated to the two other possibilities equal probable. Regardless of the mutation type, the action is mutated to any other possible action with a probability μ. For crossover, two-point crossover is applied with a probability χ. The parents stay in the population and compete with their offspring. The classifiers are inserted applying a *subsumption deletion* method. If a classifier cl exists in the population that is more general in the condition part, experienced (i.e. $exp_{cl} < \theta_{sub}$), and accurate (i.e. $\epsilon_{cl} < \epsilon_0$), then the offspring classifier is not inserted but the numerosity of the subsumer cl is increased. Finally, if the number of micro-classifiers in the population exceeds the maximal population size N, excess classifiers are deleted. A classifier is chosen for deletion with roulette wheel selection proportional to its action set size estimate as. Further, if a classifier is sufficiently experienced ($exp > \theta_{del}$) and significantly less accurate than the average fitness in the population ($f < \delta * \sum_{cl \in [P]} f_{cl} / \sum_{cl \in [P]} num_{cl}$), the probability of being selected for deletion is further increased. Note that the GA is consequently divided into a reproduction process and a separate deletion process.

Finally, *action set subsumption* may be applied. This method searches in each action set $[A]$ for the classifier that is (1) accurate, (2) experienced, and (3) most general among the ones that satisfy (1) and (2). If such a classifier exists, it subsumes all classifiers in [A] that are more specific (i.e. specify proper subsets in the condition). The more specific classifiers are deleted and the numerosity of the subsumer is increased accordingly.

3 DISTINCT PRESSURES

Although the description above explains the functioning of the system, it does not become clear why it is any good. To reveal the strength of XCS, this section analyzes the distinct evolutionary pressures separately. Section 4 reveals the interactions between the pressures.

3.1 FITNESS PRESSURE

The parameter update of prediction p, prediction error ϵ and action set size estimate as represented in formulas 1, 2, and 3 assures that the values are an average over all encountered states so far with emphasis on the recently encountered states. The fitness of a classifier is derived from its relative accuracy in $[A]$. It represents the proportional accuracy with respect to all other classifiers in $[A]$. Thus, the selection pressure is a pressure towards accurate classifiers in each environmental niche. The existence and amount of pressure towards accurate classifiers is highly dependent on problem and parameter settings. Note, that in the case when all classifiers in an action set $[A]$ are accurate or similarly inaccurate, the fitness does not directly distinguish the classifiers anymore. In this case the selection process is similar to a random selection in $[A]$. However, an experimental validation in section 5 shows that the noise in the values of unexperienced classifiers can influence the selection process.

3.2 SET PRESSURE

With respect to the population the application of reproduction in the action set results in another pressure. This pressure towards generality was stated by Wilson (1995) in his *generality hypothesis* and was later refined to an *optimality hypothesis* and further experimentally investigated by Kovacs (1997). The basic idea is that classifiers that are more often part of an action set are more often part of the GA and consequently reproduced more often as long as they are as accurate as more specific classifiers. Thus, reproduction in action sets causes an intrinsic pressure towards generality. The amount of expected (lower) specificity of classifiers in an action set is determined by the following formula:

$$s([A]) = \frac{\sum_{k=0}^{L} \binom{L}{k} (\frac{s([P])}{2})^k (1 - s([P]))^{L-k} * \frac{k}{L}}{\sum_{k=0}^{L} \binom{L}{k} (\frac{s([P])}{2})^k (1 - s([P]))^{L-k}} \quad (6)$$

Where s denotes the average proportion of specific values in the conditions of the classifiers in the referred set. Considering a specificity of $[P]$ in the population the formula determines the resulting specificity in the action set assuming a binomial specificity distribution in the population. This assumption is certainly valid in the beginning of an experiment if the population is initialized with respect to $p_\#$. In this case the average specificity will be $1 - p_\#$. It can be observed that k in the numerator and n in the denominator cause the specificity in $[A]$ to be smaller than the specificity in $[P]$. This confirms the proposition of the additional generality pressure mentioned above since the selection takes place in the action set, while deletion takes place in the more specific population. Without fitness pressure, the formula provides an estimate of the difference in specificity of selected classifiers and deleted classifiers as long as a binomial distribution is present. The above equation is enhanced in section 4.1 and experimentally validated in section 5.

3.3 MUTATION PRESSURE

In LCSs mutation appears to have a stronger impact. Generally, a random mutation process causes a tendency towards an equal number of symbols in a population. Thus, applying random mutations in a population of individuals or in particular classifiers, the result will be a population with an approximately equal proportion of zeros and ones or essentially 0, 1, and # in a classifier system. The *free mutation* described above pushes towards a distribution of $1:2$ general:specific while *niche mutation* pushes towards $1:1$. The average change in specificity between the parental classifier $s(cl(t))$ and the mutated offspring classifier $(s(cl(t+1))$ for the niche mutation case can be written as

$$\begin{gathered}\Delta_{mn} = s(cl(t+1)) - s(cl(t)) = \\ s(cl(t)) * (1-\mu) + (1 - s(cl(t)) * \mu - s(cl(t)) = \\ \mu(1 - 2s(cl(t))) \end{gathered} \quad (7)$$

and for the free mutation case as

$$\begin{gathered}\Delta_{mf} = s(cl(t+1)) - s(cl(t)) = \\ s(cl(t)) * (1-\mu/2) + (1 - s(cl(t)) * \mu - s(cl(t)) = \\ 0.5\mu(2 - 3s(cl(t))). \end{gathered} \quad (8)$$

Thus, mutation alone pushes the population towards a specificity of 0.5 and 0.66 applying niche mutation and free mutation, respectively. The strength of the pressure depends on the mutation type, the frequency of the GA application (influenced by the parameter θ_{ga}), and the probability μ of mutating an attribute.

3.4 DELETION PRESSURE

Due to its proportionate selection method with respect to the action set size estimate as and possibly the fitness F of a classifier, the deletion pressure is difficult to formalize. Generally, the selection of deletion classifiers in the population does not result in any set pressure as encountered in the selection method. Thus,

without any bias the average specificity of deleted classifiers is equal to the average specificity in the population $s([P])$.

Due to the bias towards selecting classifiers that occupy larger action sets, deletion stresses an equal distribution of classifiers in each environmental niche. The further bias towards low-fit classifiers was investigated and optimized by Kovacs (1999) in that a low fitness is only considered if the classifier has a sufficient *experience* assuring that the classifier is indeed inaccurate.

3.5 SUBSUMPTION PRESSURE

The final pressure in XCS is the pressure induced by the subsumption deletion method. Due to the experience and accuracy requirement it is assured that subsumption only applies to accurate classifiers. Once accurate classifiers are found subsumption deletion pushes towards maximal *syntactic* generality in difference to the set pressure above which only pushes towards generality if generality also assures a higher applicability rate. *GA subsumption deletion* hinders the insertion of more specific classifiers once an accurate, more general one evolved. *Action set subsumption* is much stronger since an accurate, more general classifier actually absorbs all more specific classifiers regardless if it already existed or was just generated.

To summarize, the subsumption pressure is an additional pressure towards accurate, maximally general classifiers (i.e. classifiers that are still accurate and in the mean time as general as possible). Subsumption only applies once accurate classifiers are found.

4 PRESSURE INTERACTION

After we analyzed the various evolutionary pressures separately in the last section, this section puts the pressures together and analyzes their interaction. First, the interaction of set, mutation, and deletion pressure is formulated. Next, the effect of subsumption pressure is discussed. Finally, we provide a visualization of the interaction of all the pressures. The theoretical analyzes are experimentally validated in section 5.

4.1 SPECIFICITY PRESSURE

When analyzing the interaction of set, mutation, and deletion pressure described above, we realize that all three pressures influence the average specificity in the population. Thus, the three pressures together result in a *specificity pressure* in the population.

Due to the problem dependence of the fitness pressure, we cannot formulate the pressure and consequently need to assume a similar fitness of all classifiers in our analysis. As it will be shown in section 5, this assumption holds when all classifiers are accurate and nearly holds when all are similarly inaccurate. The addition of subsumption pressure is discussed in section 4.2.

Despite the fitness irrelevance assumption, deletion pressure is also dependent on the action set size estimate *as* of a classifier. In accordance with Kovacs's insight in the relatively small influence of this dependence (Kovacs, 1999), we assume a random deletion from the population in our formulation. Thus, as stated above, a deletion results on average in the deletion of a classifier with a specificity equal to the specificity of the population $s([P])$. The generation of an offspring, on the other hand, results in the insertion of a classifier with an average specificity of $s([A]) + \Delta_{fm}$ or $s([A]) + \Delta_{nm}$ dependent on the type of mutation used. Putting the observations together, we can now calculate the average specificity of the resulting population after one learning cycle:

$$s([P(t+1)]) = \\ s([P(t)]) + f_{ga} * \frac{2 * (s([A]) + \Delta_m - s([P(t)]))}{N} \quad (9)$$

The parameter f_{ga} denotes the frequency of the GA application in XCS. The formula adds to the current specificity in the population $s([P(t)])$ the expected change in specificity calculated by the difference between the specificity of the two reproduced and mutated classifiers $s([A]) - \Delta_m$ and $s([P(t)])$. Although the frequency f_{ga} is written as a constant in the equation, it is actually dependent on $s([P(t)])$ as well as the specificity distribution in the population. Thus, f_{ga} can generally not be written as a constant. By setting θ_{ga} to one, it is possible, though, to force f_{ga} to be one since the average time since the last GA application in an action set (not generated by covering) will always be at least one.

4.2 ADDING SUBSUMPTION

Although we revealed the cause and existence of the set pressure that pushes the population towards more general classifiers once all are accurate, we also showed that this pressure is somehow limited. Equation 9 shows that without subsumption the convergence of the population towards accurate, maximally general classifiers is not assured. Essentially, if the specificity of the accurate, maximally general classifiers in a problem is lower than the value of the converged equation 9, then the population will not completely converge to those classifiers. Another reason for a lack of convergence can be that the set pressure is not present at all. This can happen, if XCS encounters only a subspace of all possible examples in the universe $\{0,1\}^L$.

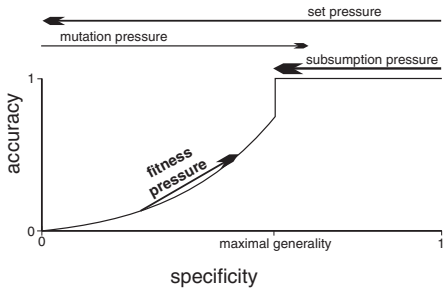

Figure 1: Together, the evolutionary pressures lead the population towards the accurate, maximally general classifiers.

In this case, the subsumption pressure results in a further convergence to the intended accurate, maximally general classifiers.

4.3 ALL PRESSURES

Finally, fitness can influence several other pressures as mentioned above. Generally, the fitness pushes from the over-general side towards accuracy as long as the environment provides helpful, layered payoff or the consistency of predicting an outcome is biased in over-general classifiers as revealed in Butz, Kovacs, Lanzi, and Wilson (2001). Thus, in terms of specificity fitness results in a pressure towards the specificity of maximally general classifiers from the over-general side. The pressures are visualized in figure 1. While set and mutation pressure (free mutation is visualized) are accuracy independent, subsumption and fitness pressure are guided by accuracy. Due to its distinct influences, deletion pressure is not visualized.

5 EXPERIMENTAL VALIDATION

In order to validate the proposed pressures and the specificity behavior formulated in equation 9 we apply XCS to boolean strings of length $L = 20$ with different settings. The following figures show runs with varying mutation from 0.02 to 0.20. In each plot the solid line denotes the formula and the dotted lines represent the XCS runs. All curves are averaged over 20 experiments. If not stated differently, the population is initially filled up with randomly generated classifiers with don't care probability $p_\# = 0.5$. Niche mutation is applied. The remaining parameters are set as follows: $N = 2000$, $\beta = 0.2$, $\alpha = 1$, $\epsilon_0 = 0.001$, $\nu = 5$, $\gamma = 0.95$, $\theta_{ga} = 1$, $\chi = 0.8$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = \infty$, and $p_{explr} = 1$.

5.1 FIXED FITNESS

In order to validate the different assumptions in the theory, we start by examining runs where the fitness influence is eliminated. That is, each time a classifier is updated, its fitness is not updated as usual but is simply set to its numerosity (completely eliminating the fitness influence in selection). The same is done in covering. Moreover, we eliminated the distinct deletion pressure influences by deleting classifiers proportionally to their numerosity num regardless of their value of as or F. This section investigates the influence caused by the two mutation types. Moreover, we investigate the influence of the GA threshold θ_{ga} as well as the influence of an initialization of the population.

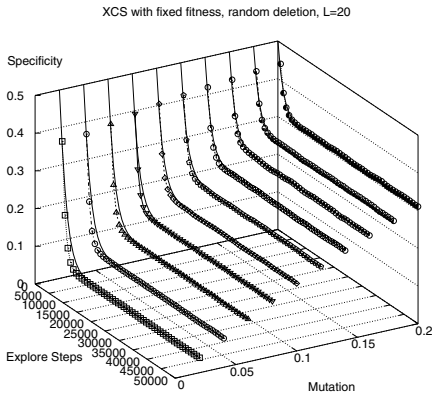

Figure 2: Eliminating the fitness influence, the specificity in XCS behaves exactly like the theory.

With the restrictions, the runs exactly match the theory as shown in figure 2. The initial specificity of 0.5 drops off quickly in the beginning due to the strong *set pressure*. However, soon the effect of mutation becomes visible and the specificity in the population converges as predicted. The higher the mutation rate μ, the stronger the influence of mutation pressure, which is manifested in the higher convergence value in the curves with higher μ.

Although the mutation pressure becomes visible in the variation of μ, figure 3 further reveals the influence caused by mutation. As formulated in equation 8 the mutation pressure is slightly higher when applying the free mutation type. When directly comparing figure 2 and figure 3 one can observe that the higher the parameter μ, the higher the difference in the mutation pressure.

As stated earlier, we set the GA threshold θ_{ga} to one to assure a GA frequency f_{ga} of one. When altering θ_{ga} setting it to the common value of 25 figure 4 reveals what has been suspected before. For equation 9 to exactly match the specificity change, f_{ga} cannot be denoted by a constant but is actually dependent on

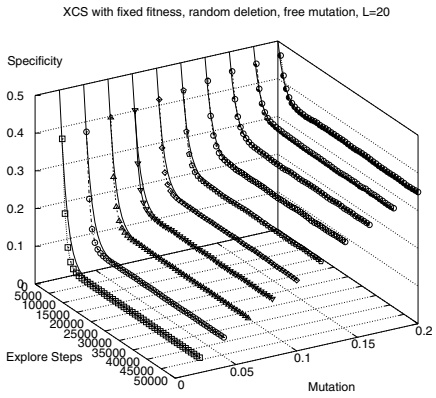

Figure 3: As predicted is the pressure caused by free mutation higher than the one by niche mutation.

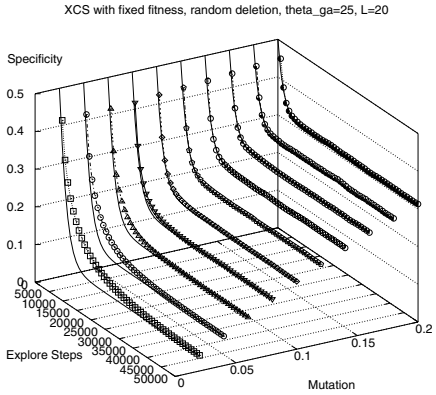

Figure 4: When increasing the threshold θ_{ga} the GA frequency and consequently the specificity pressure decreases once the specificity drops.

the current specificity in the population. Once the specificity in the population has dropped, the action set sizes increase since more classifiers match a specific state. Consequently, more classifiers take part in a GA application, more time stamps ts are updated, the average time since the last GA application in the population and in the action sets drops, and finally the GA frequency drops which is observable in the graph. However, as predicted by equation 9, despite its dependence on the actual specificity, f_{ga} does not influence the convergence value.

Although we decided to initially fill up the population with classifiers to assure a perfect binomial specificity distribution in the beginning of the run, this appeared not to be necessary as shown in figure 5. The figure shows runs in which the population is initially empty.

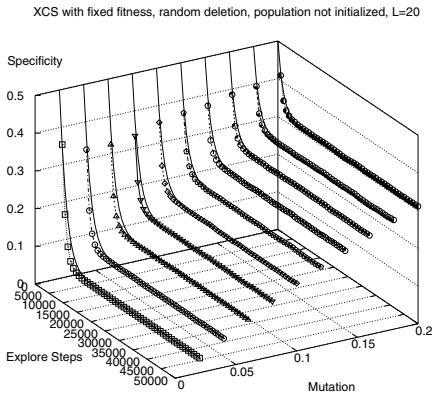

Figure 5: Without initializing the population, the generality drops slightly faster in the beginning.

The only effect observable is that the specificity drops off slightly faster in the beginning of a run. Since the population does initially not contain 2000 classifiers, the generality pressure is stronger which is also expressed in equation 9.

5.2 CONSTANT FUNCTION

While the fitness influence was intentionally eliminated above, this and the next section are dedicated to determine the actual fitness influence. This section applies XCS to a constant boolean function which always returns a reward of 1000. The result is that all classifiers are accurate since the prediction error will be zero. However, an influence could be possible due to the fitness determination according to the relative accuracy.

Figure 6 exhibits that this influence can be neglected when the random deletion method is applied as before. It shows that the assumption of a binomial distribution indeed holds later in the run or is at least not too harsh since the specificity exactly behaves as predicted.

When applying the usual deletion method in XCS, however, the behavior of the specificity changes. Figure 7 shows that the slope of the curves decreases. In the end, though, the convergence value is reached what is predicted by the theory. Since the observable influence can only be caused by the bias of the deletion method towards deleting classifiers in larger niches, the reason for this behavior follows. Since the specificity drops, the action set size increases as noted before. Thus, since more general classifiers are more often present in action sets, their action set size estimate as is more sensitive to the change in the action set size and consequently, it is larger in more general classifiers while specificity drops. Eventually, all as

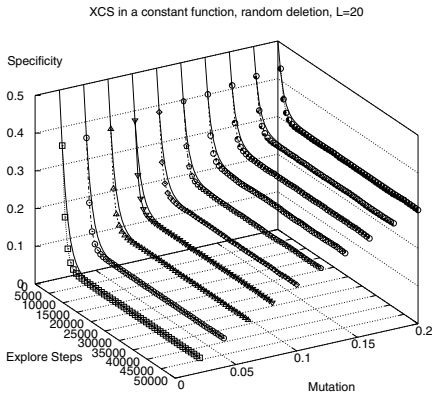

Figure 6: When applied to a constant function, the changing specificity still matches the proposed theory.

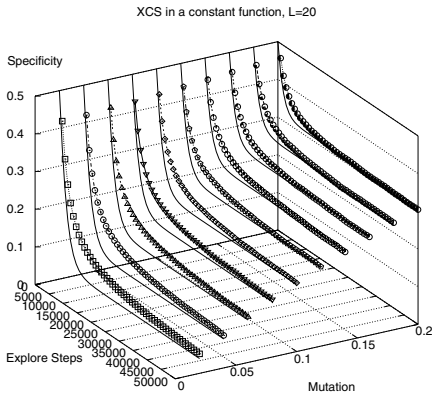

Figure 7: As expected, the effect caused by the deletion method is minor.

values will have adjusted to the change and the predicted convergence value is achieved. This proposition is further validated by the fact that the difference in the runs and the theory are smaller and become equal faster with a higher mutation rate μ since the specificity slope is not as steep as in the curves with lower μ values.

5.3 RANDOM FUNCTION

While the fitness influence remained small in the case of a constant function, much more noise is introduced when applying XCS to a random function. The final two curves reveal the behavior of XCS in a random boolean function that randomly returns rewards of 1000 and 0.

Figure 8 exhibits that in the case of a random function the fitness influences the specificity slope as well as the convergence value. The convergence takes longer and,

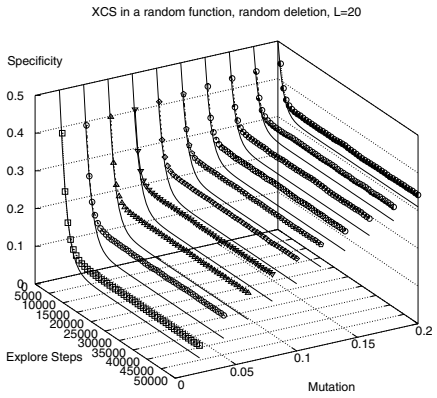

Figure 8: Applied to a random function, the specificity stays on a higher level due to the stronger noise in more specific classifiers.

moreover, the convergence value stays larger. Since again random deletion is applied, the fitness pressure is the only possible influence. Although we don't have a proof in hand we believe the following. Since the encountered rewards are 0 and 1000, the reward prediction of the classifiers fluctuates around 500 and consequently the prediction error around 500 as well. As in the case of the more sensitive action set size estimates above, here the sensitivity manifests in the prediction error ϵ. More specific classifiers have a less sensitive ϵ and consequently a higher variance in the ϵ values. This by itself does not cause any bias, however, since the accuracy calculation expressed in equation 4 scales the prediction error to the power ν, which is set to the usual value five, the higher variance causes an on average higher fitness.

When applying the normal deletion method, the deletion method causes a further increase in specificity as shown in figure 9. Since the specificities do not converge to those in figure 8, the cause must lie in the bias towards deleting experienced, low-fit classifiers. Interestingly, this pressure is strongest when μ is set to approximately 0.1. A more detailed analysis showed that this is the case because the variance in the fitness values is high as well as the more general classifiers are sufficiently experienced so that their possible low fitness value may be considered during deletion. When the specificity drops further due to a lower μ, the variance in fitness decreases significantly and the effect diminishes. On the other hand, when increasing μ further, the average experience in the population decreases under the crucial value of $\theta_{ga} = 20$ and consequently, the additional bias towards low-fit classifiers applies decreasingly often.

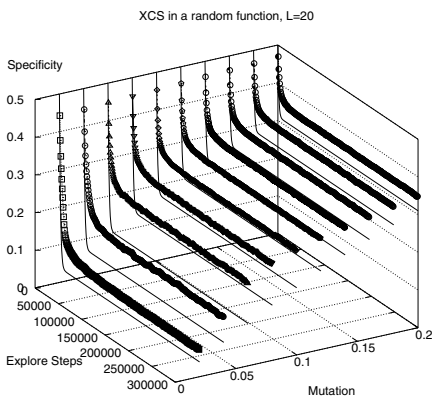

Figure 9: The fitness biased deletion method further influences the specificity.

6 SUMMARY AND CONCLUSION

This paper has investigated various evolutionary pressures in XCS. By analyzing the pressures separately and next investigating their interactions, we were able to derive a formula that can predict the specificity change in the population of XCS. While this change has been hypothesized long ago, we are now able to confirm the hypothesis mathematically and use the derived formula to explain and predict the behavior of the population in XCS. Although the fitness influence is not incorporated in the formula we showed that the formula is applicable in the case of all accurate and all similarly inaccurate. Essentially, the formula can predict how the specificity in a population will evolve once accurate but possibly over-specific classifiers are found. Moreover, it can also predict how the population evolves if there are only inaccurate classifiers and the fitness pressure towards accuracy from the over-general side is very weak.

A final important insight is that regardless of the initial specificity introduced by the *don't care probability* $P_{\#}$, we now know how the specificity changes and to which value it converges. Future research should use this insight and proceed to control the changes in specificity where necessary.

ACKNOWLEDGMENTS

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-1-0163. Research funding for this work was also provided by the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U. S. Government.

References

Butz, M. V., Kovacs, T., Lanzi, P. L., & Wilson, S. W. (2001). How XCS evolves accurate classifiers. *Submitted to the Genetic and Evolutionary Computation Conference (GECCO-2001).*

Butz, M. V., & Wilson, S. W. (2001). An algorithmic description of XCS. *In Lanzi, P. L., Stolzmann, W. and Wilson, S. W. (Eds.), Advances in Learning Classifier Systems, LNAI 1996*. to appear.

Kovacs, T. (1997). XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, & Pant (Eds.), *Soft Computing in Engineering Design and Manufacturing* (pp. 59–68). Springer-Verlag, London.

Kovacs, T. (1999). Deletion schemes for classifier systems. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)* (pp. 329–336). San Francisco, CA: Morgan Kaufmann.

Widrow, B., & Hoff, M. (1960). Adaptive switching circuits. *Western Electronic Show and Convention*, *4*, 96–104.

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

Wilson, S. W. (1998). Generalization in the XCS classifier system. In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., & Riolo, R. (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference* (pp. 665–674). San Francisco: Morgan Kaufmann.

Designing an Optimal Evolutionary Fuzzy Decision Tree for Data Mining

Hung-Ming Chen
Department of Information Science, Feng Chia University, Taichung, Taiwan, ROC.
E-mail: hmchen@ms25.url.com.tw

Shinn-Ying Ho
Department of Information Science, Feng Chia University, Taichung, Taiwan, ROC.
E-mail: syho@fcu.edu.tw

Abstract

The amount of digital data processed by computers grows extremely fast. Therefore, how to discover useful knowledge from large digital data has become a very important issue. Decision trees like ID3 and C4.5 are efficient classifiers in generating classification rules from large training patterns. Besides, decision tree classifiers can directly generate linguistic if-then rules which are human understandable knowledge. The design of an optimal evoltionary fuzzy decision tree is proposed in this paper. Fuzzy decision tree integrates the flexibility of fuzzy sets and comprehensibility of decision trees. In order to generate a more compact fuzzy decision tree, we use flexible trapezoid fuzzy sets to define membership functions. The parameters of membership functions are optimized by a powerful intelligent genetic algorithm. Furthermore, additional control genes in a chromosome are used to perform feature selection and redundant fuzzy set deletion simultaneously. The performance of the proposed fuzzy decision tree is superior to conventional decision trees and the existing fuzzy rule-based approaches in terms of both classification rate and number of fuzzy rules.

1 INTRODUCTION

The amount of digital data processed by computers grows extremely fast. In real-world applications of E-commerce, inestimable amount of transaction records are stored in databases. Therefore, how to discover useful knowledge hidden in large databases is very important.

In data mining researches, generation of association rules and classification rules is the most important issue. Generation of association rules is to find correlative attributes in large databases. If the frequency of two attributes both appear in the same record is greater than a threshold, we would assume that a relation exists between these two attributes. The number of relative attributes can be extended to represent more complex relationships. For example, many people would buy milk and bread simultaneously when they go to supermarkets. Therefore, the market managers may make special discounts according to this phenomenon.

Classification is to divide training patterns into subsets according to their attributes such that most of the patterns in the same subset belong to the same class. For example, we can analyse the personal information (age, debt, etc.) of applicants for credit cards and find some rules to classify them into an acceptance set and a rejection set. If the classification rate of these rules is accurate enough, the classifier may take the place of human checkers, which can avoid mistakes.

This paper aims at the generation of classification rules. Decision tree classifier is the most frequently used one for solving classification problems. A decision tree is composed of nodes and arcs. Non-terminal nodes represent the attributes for making decisions, arcs are attribute values, and terminal nodes represent pattern classes. The most famous and widely used decision tree classifiers are ID3 (Quinlan, 1986) and C4.5 (Quinlan, 1993). Both of them use entropy measurement based on the information theory that can generate compact decision trees.

Recently, neural networks are also become popular approaches for solving classification problems because they are easy to implement and do not require prior domain knowledge. However, the learning results of neural networks are generally hard to interpret. It would be a serious problem if we want to understand or verify the decisions. Besides, neural networks may require more training time.

Comparing decision tree classifiers with other ones for data mining, we can find several advantages of decision tree classifiers:

- Require less training time.
- Directly generate if-then linguistic rules which are more comprehensible.
- Can handle training data with noise.
- Can effectively handle high dimensional classification problems.

- Fast classification

In real world, class boundaries of data are usually non-axis-parallel. However, conventional decision tree always partitions the feature space in an axis-parallel way. Therefore, conventional decision tree may generate exceeded rules in training phase. Fig. 1 gives an example of a non-axis-parallel data distribution and the partitioning result obtained by a conventional decision tree.

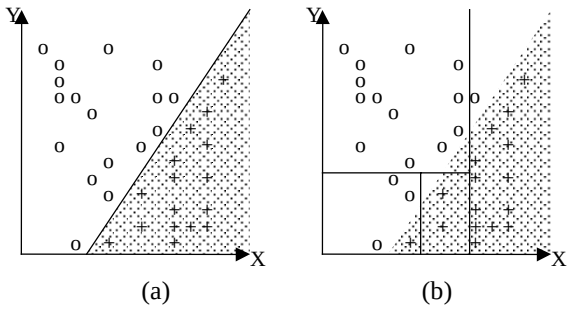

Figure 1: (a) Non-Axis-Parallel Class Boundary
(b) Partitioning Result

Furthermore, the crisp boundaries may increase the probability of misclassification in test phase. If the location of a test pattern is very close to the class boundary, the uncertainty of the classification result of this pattern will increase.

Fuzzy systems are successfully applied to increasingly numerous areas where designs are based entirely on the linguistic equivalent of human experience or knowledge. Fuzzy inference mechanism is very suitable for uncertainty handling. Besides, the class boundaries formed by fuzzy rules can be non-axis-parallel. Fig. 2 illustrates an example of fuzzy partition in a 2-D input space. Various successful fuzzy classifiers have been proposed. The fuzzy decision tree was also included.

Generally, a fuzzy system consists of fuzzy membership functions and a fuzzy rule base. The triangular membership functions are the most frequently used in fuzzy systems. The major advantage is they can reduce the number of required parameters for membership function representation. However, the flexibility of membership functions is limited. (Medasani *et al.*, 1998) have noted that membership functions must be flexible enough to develop a high-performance fuzzy classifier.

In this paper, the design of an optimal evolutionary fuzzy-ID3 decision tree is proposed, which has following the properties:

- Use flexible trapezoid fuzzy sets to define fuzzy membership functions.
- Use additional control genes to perform feature selection and redundant fuzzy set deletion simultaneously.
- Use a powerful optimization algorithm, an intelligent genetic algorithm, to solve the large parameter optimization problem associated with an optimal fuzzy decision tree design.

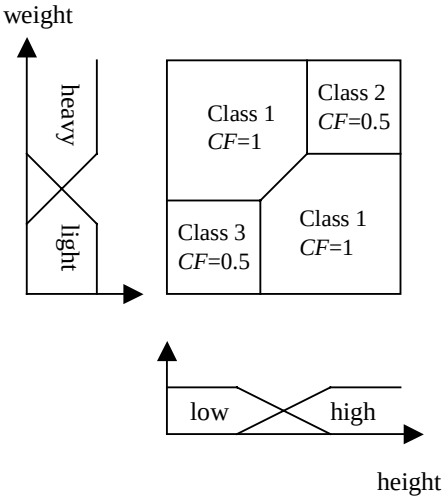

Figure 2: Non-aixs-parallel Boundaries Generated Using Fuzzy Partition

The paper is organized as follows. In next section, we briefly overview various fuzzy classifiers. In Section 3, we describe our algorithm for designing an optimal fuzzy-ID3 decision tree. Section 4 is the experimental result of our algorithm. The last section is our conclusions.

2 FUZZY CLASSIFIERS

Fuzzy systems (Zadeh, 1965) are widely used in control engineering. Recently, fuzzy theory has been successfully applied to various domains of computer science.

The major advantage of fuzzy systems is that they don't require any well-defined mathematic model of problems. The advantage increases the adaptability of fuzzy systems for various types of problems, especially for the type of problems which is difficult to define mathematic models.

Recently, various fuzzy systems for solving pattern classification problems have been proposed. The main procedure of applying fuzzy systems for classification is (1) partition feature space into subregions using fuzzy membership functions and (2) determine fuzzy rules corresponding to the subregions.

According to the partition strategy, the fuzzy partitions approaches can be categorized into three types (Yen, 1999): (1) gird partition, (2) scatter partition, and (3) tree partition.

Grid partition is most frequently used approach. It is very easy to implement. A major advantage of grid partition is that fuzzy rules obtained from fixed linguistic fuzzy grids are always linguistically interpretable. However, the number of possible rules exponentially grows with the number of input features. Therefore, it is difficult to determine the large amount of parameters of high dimensional classifier design.

Scatter partition uses multi-dimensional antecedent fuzzy sets. Rather than covering the entire input space, the method defines a subset of the input space that characterizes the fuzzy regions where training data may occur. Generally, the classifier design uses scatter partition which tries to maximize the classification rate and minimize the number of fuzzy rules and used features without taking into account the linguistic interpretability of generated fuzzy rules.

Tree partition results from a series of guillotine cuts. A guillotine cut is made entirely across the subspace to be partitioned, and each of the regions thus produced can then be subjected to independent guillotine cutting. Tree partition covers the entire feature space and the class boundaries formed by fuzzy rules can be non-axis-parallel. Tree partition can significantly relieve the problem of rule explosion and accelerate classification.

Genetic algorithms have been applied to optimize the parameters of these three types of fuzzy classifiers. In order to reduce the number of parameters in classifier design, most of the existing methods simplify the fuzzy membership functions such as using triangular fuzzy sets or symmetric fuzzy sets. However, simple fuzzy membership functions may decrease the performance of fuzzy classifiers because the search space is limited and the optimal solution may be missed.

3 OPTIMAL FUZZY DECISION TREE DESIGN

The design of an optimal fuzzy-ID3 decision tree will be presented in this section. The architecture of the proposed fuzzy decision tree could be easily extended to fuzzy-C4.5 or fuzzy-CART decision trees. In order to obtain higher classification rates, we use flexible fuzzy sets to define membership functions. Furthermore, additional control genes are used to perform feature selection and redundant fuzzy sets deletion simultaneously. Finally, we use an intelligent genetic algorithm (IGA, Ho *et al*., 1999) to solve the large parameters optimization problem of fuzzy decision tree design.

3.1 FUZZY REASONING

To determine the class of an input pattern $x_p = (x_{p1}, x_{p2}, \ldots, x_{pn})$ based on voting by multiple fuzzy if-then rules that are compatible with x_p , the general fuzzy reasoning method used in (Ishibuchi *et al.*, 1999) is adopted as follows:

$$Conf_k(x_p) = \max_{i \in R(k)} (\prod_j \mu_{U^{ij}}(x_{pj})) \cdot CF_i \qquad (1)$$

where $R(k)$ is a set of rules that classify patterns into class k, $\mu_{U^{ij}}(x_{pj})$ is the grade of fuzzy set U^{ij} at x_{pj}, and CF_i is the grade of certainty of the ith rule. The final classification result of x_p is the class k with a maximal $Conf_k(x_p)$.

3.2 FLEXIBLE MEMBERSHIP FUNCTION

In the fuzzy system, each feature has an associated fuzzy membership function. Each fuzzy membership function contains several fuzzy sets. Design of a powerful fuzzy classifier requires more flexible membership functions. Therefore, we propose a flexible fuzzy membership function using flexible trapezoid fuzzy sets. A flexible trapezoid fuzzy set can be represented by a tuple of four parameters <*Lb*, *Lt*, *Rt*, *Rb*>, as illustrated in Fig. 3.

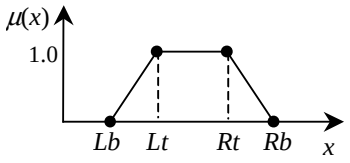

Figure 3: Trapezoid Fuzzy Set

The grade at position x for a fuzzy set can be determined using the following equation:

$$\mu(x) = \begin{cases} 0 & : x \le Lb \cup x \ge Rb \\ 1 & : Lt \le x \le Rt \\ \dfrac{x - Lb}{Lt - Lb} & : Lb < x < Lt \\ \dfrac{Rb - x}{Rb - Rt} & : Rt < x < Rb \end{cases} \qquad (2)$$

In general, a useful fuzzy membership function would contain several fuzzy sets. Fig 4 is an example of a membership function that contains three fuzzy sets.

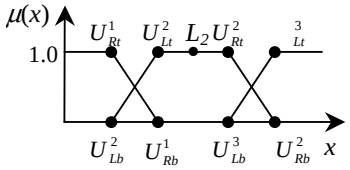

Figure 4: A Membership Function with Three Trapezoid Fuzzy Sets

For a fuzzy membership function that contains m fuzzy sets can be represented as $<U^1, U^2, \ldots, U^m>$. We assume that $U^1_{Lt} = 0$, $U^1_{Lb} = 0$, $U^m_{Rt} = 1$, and $U^m_{Rb} = 1$. It can reduce the number of parameters without losing generalization or flexibility. Therefore, IGA can be applied to determine these parameters to fit real pattern distribution. It is noted that each position on the feature axis must be covered by one fuzzy set at least.

3.3 FITNESS FUNCTION

Three objectives in designing an optimal fuzzy decision tree using IGA are as follows:

- Maximizing the number of correctly classified training patterns;
- Minimizing the number of fuzzy rules; and
- Minimizing the number of features.

Combining these three objectives, an optimization problem is formulated as the following fitness function $F(S)$:

$$\text{Maximize } F(S) = w_{NCP} \cdot NCP(S) - w_S \cdot N_r - w_F \cdot N_f \quad (3)$$

where w_{NCP}, w_S, and w_F are positive weights. $NCP(S)$ denotes the number of training patterns correctly classified by the fuzzy decision tree, N_r is the number of fuzzy rules of the fuzzy decision tree, and N_f is the number of features used in the fuzzy decision tree.

3.4 CHROMOSOME REPRESENTATION

A high performance fuzzy decision tree not only should obtain a high classification rate, but also have to minimize the number of rules. Therefore, we use additional control genes to perform feature selection and redundant fuzzy set deletion simultaneously. Since few features and fuzzy sets participate in the fuzzy decision tree construction, the generated rule base would be more compact.

In a chromosome, there are two types of genes: parametric genes and control genes. The parametric genes are applied to encode the membership functions. To make sufficient use of IGA's ability, it is useful to decrease the interaction between the correlated parameters. Therefore, the intermediate parameter L_i indicating the position of the location of a fuzzy set is introduced. Let L_1=0 and L_m=1. The parameters representing a membership function are as follows:

$$l_2, l_3, \ldots, l_{m-1}, c_1, d_1, a_2, b_2, c_2, d_2, \ldots, a_m, \text{ and } b_m.$$

All parameters are encoded as a binary string. Given a membership function that contains m fuzzy sets, the total number of parameters is equal to $5m$-6. The decoding process is as follows:

$$L_i = (1 - L_{i-1}) \cdot l_i + L_{i-1},$$
$$U^{i-1}_{Rt} = (L_i - L_{i-1}) \cdot c_{i-1} + L_{i-1},$$
$$U^{i}_{Lt} = (L_i - U^{i-1}_{Rt}) \cdot b_i + U^{i-1}_{Rt},$$
$$U^{i-1}_{Rb} = (L_i - U^{i-1}_{Rt}) \cdot d_{i-1} + U^{i-1}_{Rt},$$
$$U^{i}_{Lb} = (U^{i-1}_{Rb} - L_{i-1}) \cdot a_i + L_{i-1},$$

where i=2, 3, ..., m.

Control gene b_{Fi} is used to determine whether the feature F_i is necessary or not. Control gene b_{Sij} is used to determine whether the j^{th} fuzzy set of the membership function associated with feature F_i is redundant or not. The parameters of an entire chromosome are illustrated as Fig. 5. If b_{Fi}=0, the decoding process of membership function associated with feature F_i will be skipped, and the feature F_i will not be used in the fuzzy decision tree.

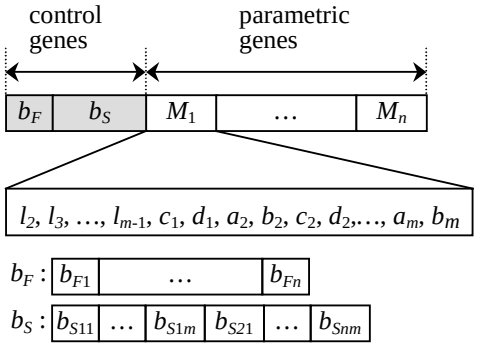

Figure 5: The Parameters of an Entire Chromosome

If b_{Sij} = 0, the j^{th} fuzzy set of the membership function associated with the feature F_i will be removed. A repairing process will also be performed to maintain the feasibility. According to the positions of removed fuzzy sets, the repairing processes are different. Fig. 6 demonstrates three different repairing results. If the number of remained fuzzy sets is equal or more than two, the parameters of fuzzy sets adjacent to the removed one should be updated (see Fig. 6(a) and 6(b)). But if the number of remained fuzzy sets is zero or one, the membership function has lost its functionality. Therefore, all the fuzzy sets would be removed and the feature will not participate in the fuzzy decision tree construction (Fig. 6 (c)).

3.5 INTELLIGENT GENETIC ALGORITHM

The fuzzy-decision tree design can be formulated as a large parameter optimization problem. Due to the huge search space, however, conventional genetic algorithms would suffer from both the low accuracy and slow convergence speed. In this paper, we use an intelligent genetic algorithm (IGA) (Ho *et al.*, 1999) which is a general-purpose large parameter optimization algorithm to optimize the parameters in the fuzzy decision tree design.

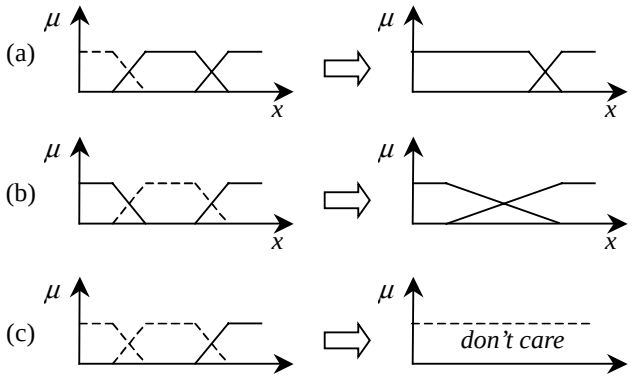

Figure 6: Different Repairing Results

The procedure of IGA is very similar to conventional genetic algorithms except the crossover operator. The IGA uses a novel intelligent crossover (IC) rather than random one-point or two-point crossovers. Fig. 7 is the procedure of IGA. The principle of IC is the evaluation of contribution of each gene/parameter based on the systematic reasoning ability of orthogonal arrays (OAs). The offspring is yielded from the best combination of genes from their parent chromosomes.

```
Initialize population Pop(0) randomly;
t = 0;
while not terminal_condition do
   evaluate each chromosome;
   select chromosomes into mating pool;
   perform intelligent crossover;
   perform mutations except the best chromosome;
   t ← t+1;
```

Figure 7: Intelligent Genetic Algorithm

An OA used in IC is described as follows. Let there be α factors with two levels (or treatments) for each factor. The total number of experiments is 2^{α} for the popular "one-factor-at-a-time" study. The columns of two factors are orthogonal when the four pairs, (1,1), (1,2), (2,1), and (2,2), occur equally frequently over all experiments. When any two factors in an experimental set are orthogonal, the set is called an OA. To establish an OA of α factors with two levels, we obtain an integer $\beta = 2^{\lceil \log_2(\alpha+1) \rceil}$, build an orthogonal array $L_{\beta}(2^{\beta-1})$ with β rows and $(\beta-1)$ columns, and use the first α columns. For instance, Table 1 shows an orthogonal array $L_8(2^7)$. The algorithm of constrcuting an orthogonal array can be found in (Leung and Wang, 2001). Orthogonal experiment design can reduce the number of experiments for factor analysis. Generally, levels 1 and 2 of a factor represent selected genes from parents 1 and 2, respectively.

Orthogonal Arrays (OAs) and factor analysis, which are representative methods of quality control (Taguchi and Konishi, 1987), also work to improve the crossover operator more efficiently. The detail procedurals of intelligent crossover are listed as follows.

Two parents breed two children using IC at a time by means of orthogonal arrays (OAs).

Table 1: $L_8(2^7)$ Orthogonal Array

	Factors							
Exp. no.	1	2	3	4	5	6	7	Function Evaluation value
1	1	1	1	1	1	1	1	y_1
2	1	1	1	2	2	2	2	y_2
3	1	2	2	1	1	2	2	y_3
4	1	2	2	2	2	1	1	y_4
5	2	1	2	1	2	1	2	y_5
6	2	1	2	2	1	2	1	y_6
7	2	2	1	1	2	2	1	y_7
8	2	2	1	2	1	1	2	y_8

Let y_t denote the positive function evaluation value of experiment number t. Let $Y_t = y_t$ $(1/y_t)$ if the objetive function is be maximized (minimized). Define the main effect of factor j with level k as S_{jk}:

$$S_{jk} = \sum_{t=1}^{\beta} Y_t^2 \cdot F_{jtk} \qquad (4)$$

where flag $F_{jtk} = 1$ if the level of experiment number t of factor j is k; otherwise, $F_{jtk} = 0$. Notably, the main efect reveals the individual effect of a factor. The most effective factor j has the largest main effect difference (MED) $|S_{j1}-S_{j2}|$. If $S_{j1} > S_{j2}$, the level 1 of factor j make a better contribution to the optimization function than level 2 does. Otherwise, level 2 is better.

The steps to use OA to achieve the IC are described as follows:

Step1: Select the first α columns of OA $L_{\beta}(2^{\beta-1})$ where $\beta = 2^{\lceil \log_2(\alpha+1) \rceil}$. Note that each parameter encoded in a chromosome is regarded as a factor in OA.

Step 2: Let levels 1 and 2 of factor j represent the j^{th} parameters of chromosomes coming from their parents 1 and 2, respectively.

Step 3: Compute the fitness value y_t for experiment number t, where $t = 1, 2, \ldots, \beta$.

Step 4: Compute the main effect S_{jk} where $j = 1, 2, \ldots, \alpha$ and $k = 1, 2$.

Step 5: Determine the best level for each parameter. Select level 1 for the j^{th} parameter if $S_{j1} > S_{j2}$. Otherwise, select level 2.

Step 6: The chromosome of the first child is formed from the best combination of the better parameter from the derived corresponding parents.

Step 7: Rank the most effective factors from rank 1 to rank α. The factor with a large MED has a higher rank.

Step 8: The chromosome of the second child is formed similarly as the first child, except that the factor with the lowest rank adopts the other level.

3.6 FUZZY DECISION TREE CONSTRUCTION

The fuzzy decision tree construction procedure is also similar to conventional decision tree, except the entropy measurement. The fuzzy entropy measurement is employed in the fuzzy decision tree construction instead of conventional entropy measurement. The entropy of the attribute F for node i, E_F^i, is defined as (5) (Kim *et al*., 1999):

$$E_F^i = \sum_j (p_{ij} I^j) \tag{5}$$

$$I^j = -\sum_{k \in C} (p_k^j \log_2 p_k^j)$$

$$p_k^j = \frac{\sum_{m \in S(k)} v_{im}}{\sum_{l \in D} v_{jl}}$$

$$p_{ij} = \frac{\sum_{l \in D} v_{jl}}{\sum_{l \in D} v_{il}}$$

$$v_{il} = \begin{cases} \prod_{f \in Z_i} \mu_f(x_{fl}) & : Z_i \neq \phi \\ 1 & : Z_i = \phi \end{cases}$$

The notations of the above symbols are as follows:

j : a child node of node i

I^j : fuzzy entropy of node j

p_k^j : the probability that node j belongs to class k

p_{ij} : the probability that patterns in node i fall into node j

v_{il} : the grade of pattern l in node i

Z_i : the set of attributes on the path from root to node i

D : the training set

C : the set of classes of training patterns

$S(k)$: the set of training patterns belonging to class k

Given flexible fuzzy membership functions obtained using IGA, the fuzzy-ID3 decision tree construction algorithm is as follows:

Step 1: Generate a root node and assign all training pattern to it.

Step 2: Determine each newly generated node i whether it is a terminal node or not as follows. If one of following three conditions is satisfied, let node i be a terminal node. Otherwise, let node i be a non-terminal node.

(1) $\frac{1}{|D|} \sum_{l \in D} v_{il} \leq \theta_s$

(2) $\frac{\sum_{m \in S(k^*)} v_{im}}{\sum_{l \in D} v_{il}} \geq \theta_d$ where $k^* = \max_{k \in C} (\sum_{m \in S(k)} v_{im})$

(3) all attributes have been used.

Let the class label of the terminal node i is k^* and its corresponding grade of certainty CF be determined using the following equation:

$$CF = \frac{\sum_{m \in S(k^*)} v_{im}}{\sum_{l \in D} v_{il}} \tag{6}$$

Step 3: For each non-terminal node i, find the best feature F^* with minimal entropy, where $E_{F^*}^i = \min_F (E_F^i)$ and $F \notin Z_i$. And then, generate a child node of node i for each fuzzy set in the membership function associated with F^*.

Step 4: If all leaf nodes are terminal nodes, end the algorithm. Otherwise, go to Step 2.

In the above algorithm, two threshold values, θ_s and θ_d, are applied to restrict the decision tree growing. This approach can prevent overfitting in fuzzy rules generation. The former forbids generating nodes with insufficient number of training patterns. The latter forbids generating nodes whose dominant class has insufficient grade of certainty. The same strategy can be found in (Kim *et al*., 1999).

4 EXPERIMENTAL RESULTS

In this section, several standard benchmark data sets are used to demonstrate the superiority of the proposed method. All of the data sets can be found in UCI machine learning databases.

Before classification, all the feature values were normalized to real numbers in the unit interval [0, 1]. In all experiments, the parameters of IGA are population size N_{pop} = 20, crossover rate P_c = 0.9, and mutation rate P_m = 0.1. The maximal generation is equal to 50. The weighted values of the fitness function are as follows: w_{NCP} = 1000, w_S = 10, and w_F = 1. The maximal number of fuzzy sets for a feature is equal to 3.

The performance of the proposed optimal fuzzy decision tree is compared with several existing methods:

(1) Simple fuzzy grid, distributed fuzzy if-then rules, CF criterion, NM criterion, and RM criterion (Ishibuchi *et al*., 1993).

(2) Fuzzy associative memory (Jang and Choi, 1996).

(3) Fuzzy-ID3 decision tree (Kim *et al*., 1999).

(4) C4.5 release 8 (Quinlan, 1993).

Table 2: The Best Case of Classification Rate and Number of Rules for Various Algorithms

Training rate	Simple fuzzy grid	Distributed fuzzy if-then rules	CF criterion	NM criterion	RM criterion	Jang and Choi	Optimal FDT algorithm
15%	91.3%(455)	92.4%(8328)	91.1%(253)	89.8%(71)	89.6%(72)	88.3%(32)	98.0%(3)
20%	92.7%(1727)	93.8%(20512)	91.8%(307)	93.0%(83)	93.3%(87)	91.6%(36)	98.0%(3)
40%	93.5%(2452)	95.4%(63069)	93.8%(307)	93.9%(105)	94.1%(107)	93.3%(46)	97.0%(3)
60%	94.5%(3440)	95.8%(140498)	95.1%(528)	94.8%(150)	94.6%(150)	95.0%(46)	98.0%(3)

4.1 IRIS CLASSIFICATION PROBLEM

The iris data comprise of 150 four-dimensional patterns from three classes. The total number of parameters in the fuzzy decision tree design is 52. Table 2 shows the testing phase performance of conventional fuzzy rules generation approaches and our optimal fuzzy decision tree. The reported results of conventional approaches are gleaned from the literature (Kim *et al*., 1999). When the training rate increases, the classification rates of all conventional fuzzy systems also increase. However, they require more fuzzy if-then rules.

In contrast to the conventional fuzzy systems, the performance of the proposed optimal fuzzy decision tree is much superior. The classification rates are always higher then conventional fuzzy systems at different training rates. Besides, the number of rules is much less than the conventional fuzzy systems and it would not increase with training rate.

Table 3 shows the performance of various decision trees, where the results of our approach are obtained from 20 independent experiments. Similar to the above experiments, the classification rates of other decision trees also increase with training rate. We can observe that the number of rules generated by decision trees is less than the conventional fuzzy systems. Comparing with other decision trees, our fuzzy decision tree can obtain higher classification rate using equal or less number of fuzzy rules.

Table 3: The Performance Comparison for Various Decision Trees

Training rate	C4.5Rules	Kim *et al*. (*best*)	Proposed optimal Fuzzy-ID3	
			(*best*)	(*avg.*)
15%	67.0% (3)	91.3% (3)	98.0% (3)	93.0% (3.75)
20%	93.3% (3)	95.3% (3)	98.0% (3)	94.0% (3.8)
40%	92.0% (3)	96.0% (3)	97.0% (3)	95.0% (3.85)
60%	95.0% (5)	96.0% (3)	98.0% (3)	96.0% (3.75)

At low training rate, the classification rate in testing phase of C4.5Rules is the worst. The major reason is that the class boundaries learning from insignificant number of training patterns are very sensitive. Though it may correctly classify the training patterns, quite amount of test patterns that are close to the boundaries may be classified incorrectly. Besides, in spite of a high training rate, we also can observe the overfitting of C4.5Rules.

4.2 WINE CLASSIFICATION PROBLEM

The wine data consist of 178 patterns with 13 continuous features from three classes. The total number of parameters in the fuzzy decision tree design is 169. For the conventional fuzzy systems such as simple grid, the number of generated fuzzy rules will exponentially grow with the number of input features. For the wine data set, if each membership function consists of three fuzzy sets, the total number of possible fuzzy rules is equal to 3^{13}. Therefore, only decision trees are used to solve such a high dimensional pattern classification problem.

Table 4 shows the performance of a crisp decision tree and the proposed optimal fuzzy decision tree. The best result of the proposed optimal fuzzy-ID3 is always superior to C4.5Rules both in terms of both classification rate and number of rules.

Table 4: Comparison of C4.5 and The Proposed Method

Training Rate	C4.5Rules	Proposed optimal Fuzzy-ID3	
		(*best*)	(*avg.*)
15%	75.2% (3)	90.0% (3)	82.0% (3.65)
20%	74.3% (4)	92.0% (3)	85.0% (3.65)
40%	91.7% (4)	95.0% (4)	89.0% (6.30)
60%	94.5% (4)	96.5% (4)	92.3% (6.90)

5 CONCLUSIONS

In this paper, we propose an optimal evolutionary fuzzy decision tree for data mining. The fuzzy decision tree can

generate non-axis-parallel boundaries, which are more adaptable for real-world data distributions. We formulate the design of a fuzzy decision tree as a large parameter optimization problem, and all parameters are optimized by a powerful intelligent genetic algorithm. In the proposed optimal fuzzy decision tree, the flexible trapezoid fuzzy membership functions can obtain high classification rates with few fuzzy rules. Furthermore, additional control genes can perform feature selection and redundant fuzzy sets deletion that both can generate more compact decision trees.

The experiment result shows that the classification rates and comprehensibility of rules both are superior to those of other fuzzy decision trees.

References

C.L. Carr (1997). Fuzzy-evolutionary systems. In Back, Fogel, and Michalewicz (Eds). *Handbook of Evolutionary Computation*. Oxford University Press

S.-Y. Ho, L.-S. Shu and H.-M. Chen (1999). Intelligent genetic algorithm with a new intelligent crossover using orthogonal arrays. *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pp. 289-296.

H. Ishibuchi, K. Nozaki, H. Tanaka (1993). Effective fuzzy partition of pattern space for classification problems. *Fuzzy Sets and Systems*, Vol. 59, pp. 295-304.

H. Ishibuchi, T. Murata, and I.B. Türksen (1997). Single-objective and two-objective genetic algorithms for selecting linguistic rules for pattern classification problems. *Fuzzy Sets and Systems*, Vol. 89, No. 2, pp. 135-150.

H. Ishibuchi, T. Nakashima, and T. Morisawa (1999). Voting in fuzzy rule-based systems for pattern classification problems. *Fuzzy Sets and Systems*, vol. 103, pp. 223-238.

D-S. Jang, H-I. Choi (1996). Automatic generation of fuzzy rules with fuzzy associative memory. *Proceedings of the ISCA 5th International Conference*, pp. 182-186.

C. Z. Janikow (1998). Fuzzy decision trees: Issues and methods. *IEEE Trans on System, Man, and Cybernetics—Part B*, Vol. 28, No. 1, pp 1-14.

M.W. Kim, J.G. Lee, and C. Min (1999). Efficient fuzzy rule generation based on fuzzy decision tree for data mining. *Proceedings of 1999 IEEE International Conference on Fuzzy System*, Vol. 3, pp. 1223-1228.

Y.-W. Leung and Y. Wang (2001). An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Trans. on Evolutionary Computation*, Vol. 5, No. 1, pp. 41-53.

S. Medasani, J. Kim, and R. Krishnapuram (1998). An overview of membership function generation techniques for pattern recognition. *International Journal of Approximate Reasoning* 19, pp. 391-417.

J. R. Quinlan (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

J. R. Quinlan (1986). Induction on decision tree. *Machine Learning*, Vol. 1, pp. 82-106.

G. Taguchi and S. Konishi (1987). *Orthogonal Arrays and Linear Graphs*. Dearbon, MI: American Supplier Institute.

K.-S. Tang, K.-F. Man, Z.-F Liu, and S. Kwong (1998). Minimal fuzzy memberships and rules using hierarchical genetic algorithms. *IEEE Transactions on Industrial Electronics*, Vol. 45, No. 1, pp.162 –169.

J. Yen (1999). Fuzzy logic-A modern perspective. *IEEE Trans. on Knowledge and Data Engineering*, Vol. 11, pp. 153-165.

L. A. Zadeh (1965). Fuzzy sets. *Informa. Contr*., Vol. 8, pp. 338-353.

Adding a Generalization Mechanism to YACS

Pierre Gérard *,**
* AnimatLab (LIP6)
8, rue du Capitaine Scott
75015 PARIS

Olivier Sigaud **
** Dassault Aviation, DGT/DPR/ESA
78, Quai Marcel Dassault
92552 St-Cloud Cedex

Abstract

A new and original trend in the Learning Classifier System (LCS) framework is focussed on *latent learning*. These new LCSs call upon classifiers with a [*condition*], an [*action*] and an [*effect*] part. In the LCS framework, the *latent learning* process is in charge of discovering classifiers which are able to anticipate accurately the consequences of actions under some conditions. Accordingly, this process builds a model of the dynamics of the environment. This paper describes how YACS performs latent learning, and how it is enhanced by a dedicated generalization process which offers an alternative to Genetic Algorithms.

1 INTRODUCTION

Holland [Hol76] presented the first ideas about LCSs (Learning Classifier Systems). The capability of generalizing is the main advantage of LCSs with respect to other reinforcement learning systems like *Q-learning* [Wat89]. It allows to consider several perceived situations within a common description so that the representation of the problem gets smaller. The accuracy based approach in Wilson's XCS [Wil95] overcomes the problem in previous LCSs where especially deferred reward leads to over-generalization.

Another concern in the general reinforcement learning framework is to build an internal model of the dynamics of the environment. This model can be used to adapt the policy further and faster. In multi-step problems, an agent can learn to *anticipate* what happens immediately after the execution of an action. This learning process can take place even in the absence of reward. Such a model of the dynamics of the environment can be learned *latently* and allows lookahead mechanisms. In order to use LCSs to learn a model of the dynamics of the environment, Holland [Hol90] proposed an implicit approach based on tagged internal messages. Riolo [Rio91] implemented this idea in his CFSC2 and demonstrated its latent learning capability. A more explicit linkage is used in CXCS [TB00].

In contrast with all these approaches, ACS (Anticipatory Classifier System, [Sto98]) and YACS (Yet Another Classifier System, [GSS01]) both form $C-A-E$[1] classifiers. This formalism is similar to Sutton's DynaQ+ [Sut91] approach but draws benefits of the generalization capability of LCSs. ACS and YACS both take advantage of the information provided by the succession of situations in order to drive the classifier discovering process. Therefore, they use heuristics instead of Genetic Algorithms, which are general but not explicitly driven by experience. This way, YACS explores the solution space rationally, so as to be able to tackle large problems like the Sheep-dog problem described in [SG01].

In [GSS01], we showed how the latent learning process in YACS leads to near-optimal but not optimal representations of the dynamics of the environment. This paper focuses on the *latent learning* process of YACS and presents the generalization process which overcomes the near-optimality problem.

In section 2 we show how the formalism used in YACS allows generalization. In section 3 we briefly describe the heuristics used for the *latent learning* process in YACS. For further details or a comparison with ACS, please refer to [GSS01]. In section 4 we describe how we introduce a generalization process in YACS. In section 5 we show experimentally how this new process helps to overcome the over-specialization problems in YACS.

[1] C stands for [*condition*], A for [*action*] and E for [$effect$]

2 GENERALIZATION IN YACS

As ACS [Sto98], YACS deals with C-A-E classifiers [2]. C parts take advantage of generality and may match several perceived situations. An A part specifies a particular action possible in the environment.

A situation is divided into several features representing perceivable properties of the environment. A C part has the same structure but it may contain *don't care* symbols "#". The E part stores for each perceived feature the expected changes in the environment when the action of the classifier is chosen and when the perceived situation matches its condition. The E part might contain *don't change* symbols "#". A *don't change* symbol in the E part means *"the feature of the perceived situation corresponding to the* don't change *symbol remains unchanged"*.

This formalism allows the classifiers to represent regularities in the environment like for instance *"In a maze, when the agent perceives a wall on north, whatever the other features are, moving north will drive the agent to hit the wall, and no change will be perceived"*

In YACS, generalization is allowed by the joint use of *don't care* and *don't change* symbols. As ACS, YACS generalizes over the anticipation of an expected effect in terms of situations, and not over the prediction of a payoff, as in XCS [Wil95].

Thus, what we call *generalization* in YACS is not the same as the generalization studied in XCS by [Lan97] for instance. As a result, it does not make sense to store information about the expected payoff in the classifiers. The list of classifiers only models the transitions in the environment.

As we showed in [GSS01], so as to perform reinforcement learning, YACS must deal with information about specific situations. So, this system uses a set P of every perceived situation encountered during the lifetime of the agent. This set only contains one single instance of each already perceived situation. Each situation is valued by the expected payoff when reaching the considered situation.

This set only contains the actually perceived situations, not all the virtually possible situations resulting from the number of features and the number of values they can take. In a large problem like the multi-agent Sheep-dog problem described in [SG01] for instance, the number of actually encountered situations is 290 while the number of virtually possible situations is 8192.

[2] C stands for [*condition*], A for [*action*] and E for [*effect*]

A way to reduce the size of this set could be to provide to YACS with a dedicated generalization mechanism which relies on the expected payoff.

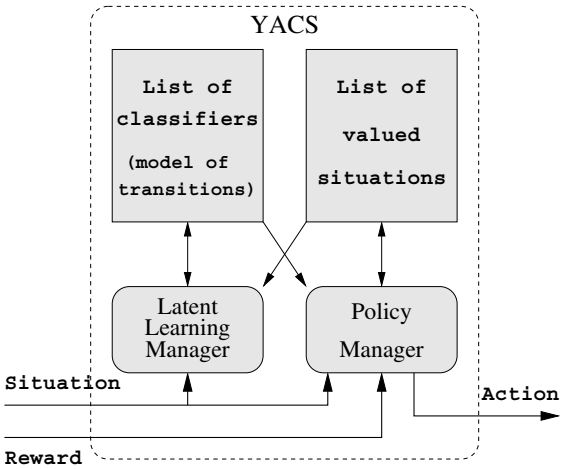

Figure 1: The YACS architecture

So, as shown in figure 1, YACS consists in several parts:

- a latent learning manager which updates the classifiers list;
- a policy manager which is in charge of updating a set of valued encountered situations. The policy manager is also in charge of selecting actions.

3 LATENT LEARNING IN YACS

The *latent learning* process is in charge of discovering $C-A-E$ classifiers with maximally general C parts that accurately model the dynamics of the environment. Unlike ACS, it learns C and E parts separately. So as to discover accurate C and E parts, YACS associates additional information to the classifiers[3]. As a result, a classifier in YACS needs more memory, but this information is used in order to reduce the complexity of the resulting model in terms of number of classifiers.

In the following sections, we briefly give the main mechanisms of the *latent learning* process as it were described in [GSS01]. For further details, please refer to this paper.

[3] two situations, a finite set of booleans markers and two sets estimates which are real numbers.

3.1 EFFECT COVERING

The effect covering mechanism is the part of the latent learning process is in charge of discovering accurate E parts (*i.e.* E parts representing actual effects of actions under some conditions). When the system learns accurate effects, it creates new classifiers with suitable E parts settled according to experience, by direct comparison of successive perceived situations.

This mechanism causes major problems in noisy environments. In such environments, it may create a lot of classifiers. Thus, we work on a new version of YACS without the effect covering mechanism.

During the *effect covering* process, YACS also updates a trace T of *good* and *bad* markers memorizing past anticipation mistakes and successes of each classifier. This trace works as a FIFO list with a finite length m.

3.2 SELECTION OF ACCURATE CLASSIFIERS

As YACS tries to build a set of classifiers that anticipate accurately, it has a deletion mechanism to remove inaccurate classifiers. The trace T of *good* and *bad* markers allows to check the anticipation abilities of a classifier.

If the trace T of a classifier is full and if it only contains *bad* markers, then YACS assumes that the classifier always anticipates incorrectly and removes it. If the trace is full and if it contains *good* and *bad* markers, we say that the classifier *oscillates* because its condition is too general. In this case, the condition must be further specialized.

3.3 SPECIALIZATION OF CONDITIONS

A C part should be as general as possible in order to represent regularities in the environment. But it must be specific enough so that the classifier does not oscillate. The specialization process incrementally specializes C parts so as to reach the right level of generality.

The classifier discovery problem is usually solved by a Genetic Algorithm. But the genetic operators do not explicitly take advantage of the experience of the agent.

YACS starts without making any distinction between situations, and incrementally introduces experience driven specializations in C parts. It uses neither mutation nor crossover operators.

The specialization process of YACS uses the *mutspec* operator introduced by [Dor94]. This operator selects a general feature of the C part[4] of an *oscillating* classifier, and produces one new classifier for each possible specific value of the selected feature. YACS improves the selection of the features to specialize by using the *expected improvement by specialization* estimate i_s associated to each *don't care* symbol in the C part of each classifier. This value estimates how much the specialization of the token would help to split the situation set covered by the C part into several sub-sets of equal cardinality.

4 GENERALIZATION OF CONDITIONS

In section 3.3 we have presented how YACS specializes C parts so as to allow the E part to be accurate. But even if this process is cautious, it may produce classifiers with a C part at a sub-optimal level of generality, in particular when YACS specializes C parts while it did not experience many possible situations.

As the specialization process, the generalization uses heuristics in order to take advantage of experience to drive the process. Thus the YACS approach differs from ACS since its generalization process does not use Genetic Algorithms [BGS00a]. This process considers sets of classifiers with the same C and A parts and decides how to specialize C parts. The generalization process also uses estimates to drive the generalization process: the *expected improvement by generalization*.

4.1 THE *EXPECTED IMPROVEMENT BY GENERALIZATION* ESTIMATES

An *expected improvement by generalization* i_g estimate is associated to each specialized feature of a C part. It estimates if the E part of the classifier would remain accurate if the considered feature was general.

At each time step, YACS knows the current situation S_t resulting from the action A_{t-1} in the situation S_{t-1}. This information is used to compute the desired effect DE which is the E part of a classifier which could have been fired at the preceding time step, and whose E part accurately reflects the changes actually perceived in the environment.

In the effect covering process, every classifier whose C part matches S_{t-1} and whose A part matches A_{t-1} is checked. In order to compute the i_g estimates, YACS checks every classifier whose A part matches A_{t-1} and whose C part does *not* match S_{t-1}.

Considering such classifiers, for each specialized fea-

[4] a feature with a *don't care* symbol

ture of the C part, YACS checks if the C part of the classifier would match S_{t-1} if the considered feature were general. In this case, the considered i_g estimate is updated:

- If the E part of the classifier equals the desired effect DE, then a classifier with a more general C part would have an accurate E part and the considered i_g estimate is increased.
- If the E part of the classifier does not equal the desired effect DE, then a classifier with a more general C part would have an inaccurate E part and the corresponding i_g estimate is decreased.

The i_g estimates are increased and decreased according to a Widrow-Hoff delta rule. The initial values are 0.5. A general feature is given an i_g value of 0.5.

Up to that point, with this mechanism, YACS is able to check if a feature of a C part should be generalized or not.

4.2 THE GENERALIZATION PROCESS

The *expected improvement by generalization* estimates detailed above allow the classifier generalization mechanism to be driven by experience and are used in the C parts generalization process.

From one situation S_{t-1} to a new one S_t, the selected action A_{t-1} leads to some effects DE in the environment. Each time step, YACS checks if there is some possible generalization between the C parts of the classifiers such that their E part equals DE and their A part matches A_{t-1}. So, the generalization process considers sets of classifiers with the same A and E parts.

With such a set of classifiers, YACS builds a new set of classifiers which are more general or equal to the original ones. These new classifiers are such that they do not match situations already matched by classifiers with a different E part. The classifiers of the new set replace the original ones in the Classifier System.

If every estimate i_g of a classifier is lower than 0.5, it is not a good candidate for the generalization and it is added without modifications in the new set of classifiers.

In either case, a new classifier is created. A feature of the C part is generalized if its associated estimate i_g equals to the greatest among the estimates associated to the considered classifier.

The new C part may lead to a conflict with other classifiers with the same A part but a different E part. In this case, YACS does not add the new and general classifier to the new set, but the original one. A conflict is detected when two classifiers share the same A parts but have different E parts, and if at least one situation is matched by both C parts. YACS finds the possible situations in the set P of every perceived situation encountered during the lifetime of the agent (see section 2).

At this point YACS has computed a new set of classifiers such that each classifier is more general or equal to the original one, and such that none of them drives to a conflict with other classifiers in the system. The next step is to select the more general classifiers.

To do so, YACS checks iteratively every possible pair of classifiers. When the C parts of two classifiers are matching, the classifier with the smallest number of general features is removed. So the classifiers of the resulting set are not redundant.

This process allows to replace several classifiers with a smaller or equal number of classifiers. The C part of the new classifiers cover the same situations. Thus they do not drive to a conflict with other classifiers in the system.

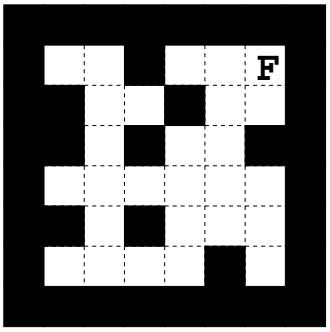

Figure 2: The Maze4 environment

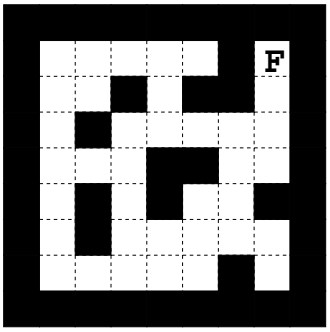

Figure 3: The Maze6 environment

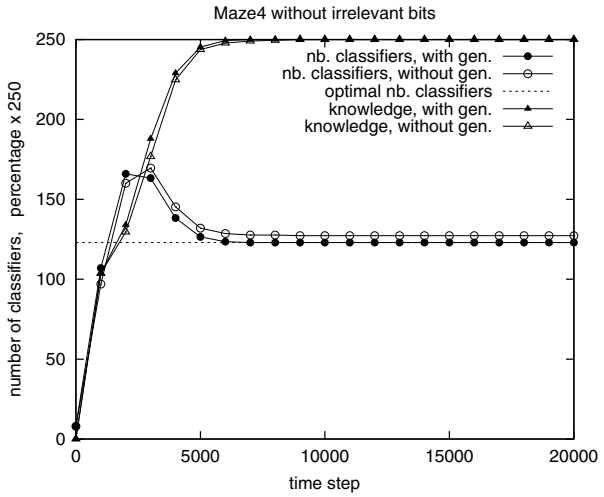

Figure 4: Evolution of the number of classifiers in Maze4

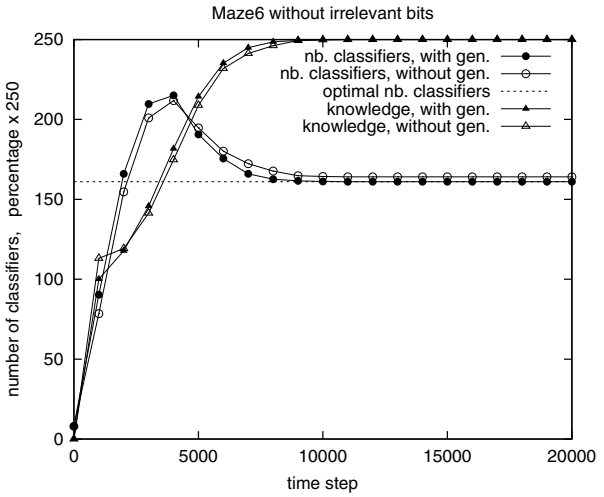

Figure 5: Evolution of the number of classifiers in Maze6

5 EXPERIMENTAL RESULTS

This section presents experimental results of YACS modeling *Wilson's woods* environments. The simulated woods environments are described in section 5.1. We show in section 5.2 how the generalization process helps the *latent learning* process of YACS to converge to the optimal number of accurate classifiers. Therefore, YACS did not learn a policy, but only a model of dynamics of the environment while moving randomly in the mazes.

5.1 THE MAZE4 AND MAZE6 WOODS ENVIRONMENTS

In woods environments, the agent is situated in a maze cell and perceives the eight adjacent cells. A cell can either be empty, or contain an obstacle ■ or food F. It can move towards any of these cells. If the agent moves towards an obstacle, it remains in the same cell.

Maze4 and Maze6 (see figures 2 and 3) have been earlier investigated by Lanzi. The experiments we present in this paper involve YACS interacting with these environments. The experiments are divided into trials. The agent starts a trial in a free cell chosen randomly. A trial ends when the agent reaches the cell with food. In that case the agent gets a reward, it gets a new perceived situation, and a new trial starts.

In these environments, it is possible to generalize the transitions which do not lead to any change. This is the case when an action leads the agent to hit an obstacle. There are respectively 93 and 135 transitions of that kind in Maze4 and Maze6 By taking advantage of generality, the transitions resulting from such actions can be modeled with 8 classifiers: one classifier for each possible action, by paying attention to the presence of a block in the direction corresponding to the action. There are no other useful regularities in Maze4 and Maze6. Since the total number of possible transitions is 208 in Maze4 and 288 in Maze6, the optimal numbers of classifiers YACS should reach are respectively 123 $(208-93+8)$ and 161 $(288-135+8)$ for Maze4 and Maze6.

Moreover, so as to provide YACS more occasions to take advantage of generalization, we add *irrelevant bits* to the perceived situations. These attributes are randomly set between 0 and 1 when a new trial starts and keep the same value during the whole trial. For a system without any generalization capability this would result in new perceived situations. But as the added perceived features are irrelevant to distinguish between situations, the optimal number of classifiers remains the same when irrelevant bits are added.

5.2 EXPERIMENTS IN MAZE4 AND MAZE6

In order to estimate the evolution of the accuracy of the model over successive time steps, we use a measure of the *percentage of knowledge* provided by the model. For each possible transition in the environment, we check if the classifier system is able to model accurately the transition. The percentage of knowledge is the ratio of possible transitions covered by *reliable* classifiers only. The memory size m is set to 5 and

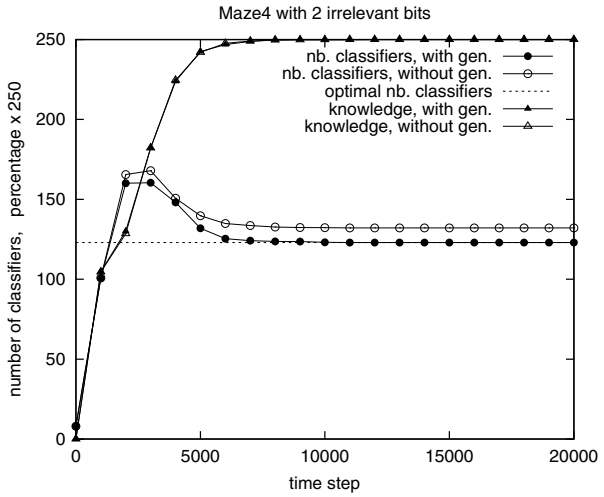

Figure 6: Evolution of the number of classifiers in Maze4 with 2 irrelevant bits

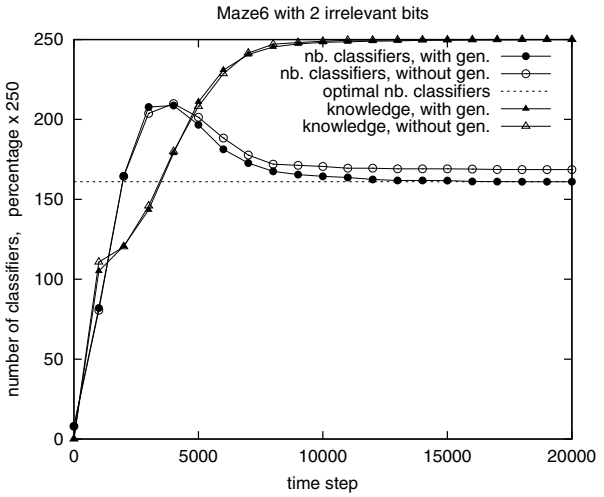

Figure 7: Evolution of the number of classifiers in Maze6 with 2 irrelevant bits

the learning rates are set to 0.1. All the results are averaged over 10 experiments.

Figure 4 presents the evolution of both the number of classifiers and the percentage of knowledge for the Maze4 experiments with 0 irrelevant features. The experimental results are shown for YACS running with and without the generalization process. Figure 5 shows the same information for the experiments with the Maze6 environment. Without generalization, the average number of classifiers discovered by YACS converges towards 127.3 (4.3 more than optimum) for Maze4 and 164.1 (3.1 more than optimum) for Maze6. This number is only near-optimal and with the generalization process enabled, the number of classifiers converges to the optimum (123 for Maze4 and 161 for Maze6, see section 5.1). Even if in Maze6, there are around 40% more transitions to model than in Maze4, these figures show that YACS does not need much more time to converge towards an optimal model of the dynamics of the environment.

During the first part of the learning process, YACS mostly creates new classifiers and their number is growing. During the second part, because the actions are selected at random, YACS may take time to experiment every possible transition as many times as necessary to remove inaccurate classifiers. As the memory size m is reduced, YACS converges faster because it takes less time to remove inaccurate classifiers, but the maximum number of classifiers during the learning process gets higher. So as to speed up the convergence towards an optimal model, we could use exploration bonuses as in [SB98]. The benefits drawn would be larger as the environment is more complex.

Figure 6 presents the evolution of both the number of classifiers and the percentage of knowledge for the Maze4 experiments when 2 irrelevant bits are added. Figure 7 shows the same in the Maze6 environment.

Without generalization, the average number of classifiers discovered by YACS converges towards 132.1 for Maze4 (9.1 more than optimum) and 168.6 for Maze6 (7.6 more than optimum). With the generalization process enabled, the number of classifiers converges to the optimum. Without generalization, the difference between the number of discovered classifiers and the optimum is greater with irrelevant bits. In this case, YACS sometimes specializes according to these bits because the estimates are not absolutely reliable, especially in the case of partial exploration of the situation space.

The results with generalization show that this process is able to reconsider early specialization mistakes without modifying a lot the learning speed. This way, YACS models the environment with a smaller number of classifiers than ACS [BGS00b] does.

6 CONCLUSION AND FUTURE WORK

The latent learning process builds a model of the dynamics of the environment even in the absence of rewards. It models how the actions modify the perceived situations. This modeling process uses information about successive perceived situations. The informa-

tion used is available at each time step. So, latent learning systems make an intensive use of the perceptual feedback offered by the sensori-motor loop. Thus, they can quickly identify relevant and general classifiers without using Genetic Algorithms.

In this paper, we briefly described the main mechanisms of the *latent learning* in YACS and we proposed a new way for performing generalization. We have shown experimentally that this additional process is able to overcome the over-specialization problems occurring in previous versions of YACS.

However, YACS is still bounded to deterministic Markov problems. In a middle term, YACS should be enhanced to tackle non-Markov problems. Moreover, we will explore in a short term a new formalism which allows to express more regularities of the environment.

References

[BGS00a] M. V. Butz, D. E. Goldberg, and W. Stolzmann. Introducing a genetic generalization pressure to the Anticipatory Classifier System part i: Theoretical approach. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference (GECCO 2000)*, 2000.

[BGS00b] M. V. Butz, D. E. Goldberg, and W. Stolzmann. Introducing a genetic generalization pressure to the Anticipatory Classifier System part ii: Experimental results. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference (GECCO 2000)*, 2000.

[Dor94] M. Dorigo. Genetic and non-genetic operators in ALECSYS. *Evolutionary Computation*, 1(2):151–164, 1994.

[GSS01] P. Gérard, W. Stolzmann, and O. Sigaud. YACS : a new Learning Classifier System using Anticipation. *Journal of Soft Computing : Special Issue on Learning Classifier Systems*, (to appear) 2001.

[Hol76] J.H. Holland. Adaptation. *Progress in theorical biology*, 1976.

[Hol90] J.H. Holland. Concerning the emergence of tag mediated lookahead in Classifier Systems. *Special Issue of Physica D*, 42:188–201, 1990.

[Lan97] P. L. Lanzi. A study of the generalization capabilities of XSC. In T. Baeck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 418–425, San Franciso, California, 1997. Morgan Kaufmann.

[Rio91] R. L. Riolo. Lookahead planning and latent learning in a Classifier System. In J.-A. Meyer and S. W. Wilson, editors, *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptative Behavior*, pages 316–326. MIT Press, 1991.

[SB98] R. S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[SG01] O. Sigaud and P. Gérard. Being Reactive by Exchanging Roles: an Empirical Study. In M. Hannebauer, J. Wendler, and E. Pagello, editors, *LNCS : Balancing reactivity and Social Deliberation in Multiagent Systems*. Springer-Verlag, (to appear) 2001.

[Sto98] W. Stolzmann. Anticipatory Classifier Systems. 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, editors, *Genetic Programming*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998.

[Sut91] R.S. Sutton. Reinforcement learning architectures for animats. In J.-A. Meyer and S. W. Wilson, editors, *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptative Behavior*, Cambridge, MA, 1991. MIT Press.

[TB00] A. Tomlinson and L. Bull. CXCS. In P.L. Lanzi, W. Stolzmann, and S.W. Wilson, editors, *Learning Classifier Systems: from Foundations to Applications*, pages 194–208. Springer-Verlag, Heidelberg, 2000.

[Wat89] C. J. Watkins. *Learning with delayed rewards*. PhD thesis, Psychology Department, University of Cambridge, England, 1989.

[Wil95] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

Mining Interesting Knowledge from Data with the XCS Classifier System

Pier Luca Lanzi
Artificial Intelligence and Robotics Laboratory
Dipartimento di Elettronica e Informazione
Politecnico di Milano
pierluca.lanzi@polimi.it

Abstract

We apply a version of XCS which exploits a general purpose representation to the problem of mining knowledge from some well-known classification tasks involving synthetic and real-world data. We show that XCS can extract interesting knowledge from data both (i) in terms of predictive accuracy on unseen cases and (ii) in terms of explicit knowledge on the phenomena described in the data. In particular, in synthetic tasks, XCS's predictive accuracy is at least as good as that of more traditional classification algorithms while it can extract rules which give an explicit insight of the data. In real world tasks, XCS outperforms C4.5 in one important medical datasets involving numerical data while it performs quite the same as C4.5 on another real world dataset involving symbolic data.

1 INTRODUCTION

Data Mining deals with the process of discovering *interesting knowledge* from large databases. Among the various knowledge discovery applications, supervised classification is probably one of the most frequent. This is defined as the problem of extracting a compact, accurate, and general description of a target phenomenon (or *concept*) represented in the data. The target phenomenon is described by a set of *examples* provided by a supervisor. Examples are described by a set of *attributes*; one particular attribute, called *class* attribute, represents the target concept. The goal of the supervised classification process is to develop a *model* of the target phenomenon described by the class attribute. This model can be subsequently used to predict the value of the class attribute of (i.e., to classify) previously *unseen* examples.

The classification model extracted from data can be represented in many ways, e.g.: neural networks, decision trees or decision rules [12]. Among the others, decision rules are one of the most important and accepted means of describing the results of the supervised classification process. Decision rules provide *explicit* descriptions of the knowledge extracted from data. Therefore, they can be used both to *predict* the class of *unseen* cases or to *get insight* of the target phenomenon by inspecting the classification model developed. Decision rules can be produced in many ways. Traditional machine learning methods *derive* rules by exploring sets of examples by means of statistical or information theoretic techniques (e.g., C4.5 [12]). Alternatively, rules can be *discovered* through methods of evolutionary computation such as genetic algorithms and learning classifier systems [8].

In the past, learning classifier systems have been sometimes applied to supervised classification problems [7]. However, during the last five years many positive results have been reported for this application area. For instance, Holmes [3] introduced EpiCS a learning classifier system specific for epidemiological data and successfully applied it to many medical domains. Wilson [15] introduced XCS an innovative (general purpose) learning classifier system model. XCS appears to be well suited for knowledge discovery tasks since, as widely shown in the literature, it is able to evolve *minimal*, accurate, and *maximally general* models of the learned task. In particular, Saxon and Barry [14] applied XCS to the Monk's problems [2] a well known *synthetic* testbed for classification algorithms. Their results showed that XCS's performance was at least as good as traditional Machine Learning techniques. Wilson [16] extended previous results and applied a version of XCS extended for integer inputs (XCSI) to a *real-world* problem: the Wisconsin Breast Can-

cer dataset (briefly WBC). Wilson's results demonstrate that also XCSI performs at least as well as state-of-the-art classification algorithms.

In this paper we want to extend those results and apply a version of XCS which exploits a general purpose representation of classifier conditions, i.e., Lisp s-expressions, to a set of well known synthetic/real-world datasets. We show that in classification problems involving synthetic data XCS with Lisp s-expression, briefly XCSL, performs at least as better as most classification algorithms. We extend these results and apply XCSL to the Wisconsin Breast Cancer dataset (WBC) and on the Voting-Record dataset. The former is described only by numerical attributes and has few missing attribute values while the latter has only attributes with symbolic values but many missing values. The results we present show that XCSL *outperforms* C4.5 on the WBC dataset and that the difference in performance *is statistically significant*. While on the Voting-Record dataset XCSL performs slightly worse than C4.5 but this difference *is not* statistically significant.

```
<condition> ::=
   "(" "NOT" <condition> ")" |
   "(" "AND" <condition> <condition> ")" |
   "(" "OR"  <condition> <condition> ")" |
   <expression>
```

(a)

```
<expression> ::=
   "(" "EQ" <value> <value> ")" |
   "(" "GT" <value> <value> ")" ;

<value> ::=
   "(" <attribute> ")" | "(" <constant> ")" ;

<attribute> ::=
   "A1" | "A2" | "A3" | "A4" | "A5" | "A6" ;
<constant> ::=
   "0" | ... | "10" ;
```

(b)

Figure 1: (a) The BNF grammar that generates the overall structure of classifier conditions and (b) the section specific for the Monk's problem (Section 4). Non-terminal symbols are in square brackets. Terminal symbols are in quotation marks.

2 DESCRIPTION OF XCSL

XCSL is a version of Wilson's XCS [15] in which classifier conditions are represented by Lisp-like s-expressions. It was introduced in [5] where some initial, promising, results were discussed. Recently, we improved XCSL [6] and applied to a wide set of problems. While in this section we briefly overview XCSL we refer the interest reader to [1, 5, 6] for further details.

XCSL works basically like all the other XCS models [15, 16] but it differs from them (i) in the covering, (ii) in the matching, and (iii) in the genetic operators.

Representation. Classifier conditions in XCSL are generated by composing the basic Boolean functions (*and*, *or*, and *not*) with a set of *elementary conditions* which test the values of system inputs and therefore depend on the problem (see [5]). For instance, in the supervised classification problems tackled in this paper, *elementary conditions* express relations among attribute values.

More formally, in XCSL, classifier conditions are specified by the BNF grammar depicted in Figure 1a which states that classifier conditions (identified by the non-terminal symbol <condition>) are generated by composing the logical *and*, *or*, and *not* functions (identified by the terminal symbols AND, OR, and NOT) with atomic expressions (identified by the non-terminal symbol <expression>) which test the values of the system inputs. To define a representation for a given problem *only* the BNF of the non-terminal symbol <expression> has to be specified. For example, in the first classification task discussed in this paper (the Monk's problem in Section 4), data are represented by six integer attributes $(a_1, a_2, \ldots, a_6)$. Accordingly, the non-terminal <expression> is defined by the piece of BNF grammar depicted in Figure 1b. This specifies that an atomic condition tests either whether two values are equal (terminal symbol EQ) or whether the first value is greater than the second one (terminal symbol GT). A value (non-terminal symbol <value> can be either one of the six attributes which define the data (i.e., <attribute>) or an integer constant (i.e., <constant>).

Matching. XCSL's inputs are represented as strings of attribute-value pairs. For instance, the string "(A1 2)(A3 1)(A4 2)" means that the current input consist of three attributes (A1, A3, and A4) whose values are 2, 1, and 2 respectively. Given an input configuration, conditions are evaluated as LISP s-expressions in which the terminal symbols, corresponding to the data attributes, are replaced with their actual values. For instance, given the former input string, the condition "(AND (EQ A1 2) (GT A3 A4))" is evaluated as "(AND (EQ 2 2) (GT 1 2))" and therefore is evaluated as false, i.e., the condition *does not* match the

former input string. If the value of an attribute is not specified in the input string (i.e., the attribute value is *missing*) all the atomic expressions in which the attribute appears are evaluated as true, e.g.: the condition "`(EQ A2 A1)`" matches the former input string since the value of `A2` is *missing*. Boolean operators are evaluated as usual [5].

Covering. The covering operator creates a classifier with a random condition that matches the current sensors and a random action. The random condition is an *or* of *three* expressions, each one matching the current input string. One expression is built as an *and* of atomic conditions so as to match *exactly* the current attribute configuration. The other two expressions are built so as to match a randomly selected *subset* of current input attributes. Missing attribute values are not considered during covering. This covering policy guarantees that all the input attribute values are matched by the first expression while it introduces generalization in the remaining two expressions.

Genetic Algorithm. As in XCS, the genetic algorithm of XCSL selects two classifiers from the action set with probability proportional to their fitnesses, copies them, with probability χ performs crossover, and with probability μ mutates them. In XCSL crossover and mutation works as in traditional Genetic Programming [4].

Condensation. In [6] we showed that, in contrast to XCS, XCSL tends to evolve populations of many overlapping classifiers. This *bloat* in the classifier population (due to the symbolic representation [6]) in some cases influences the XCSL's learning performance (see [5, 6] for details). But most important (for the application tackled in this paper) this population *bloat* makes almost impossible to study the final solutions evolved by XCSL, i.e., it makes almost impossible to analyze what kind of classification model XCSL developed.

To allow the analysis of the solutions evolved by XCSL [6] added a final *condensation* phase. Condensation [15] extracts a minimal subset of classifiers which represent the final solution. It consists of running the genetic algorithm with crossover and mutation turned off. During condensation, no new classifiers are created, fitter classifiers are reproduced preferentially and weaker ones are removed.

Note that condensation *does not* simply extract the best classifiers from the population! Condensation extracts a *compact solution* from a population which is usually made of *many*, *overlapping*, classifiers.

Dataset	N_A	N_T	N_t	N_C	$N_?$
Monks-1	6	124	432	2	0
Monks-2	6	169	432	2	0
Monks-3	6	122	432	2	0
WBC	9	286	-	2	16
Voting-Record	16	435	-	2	203

Table 1: The datasets used in this paper: N_A is the number of attributes; N_T is the number of training examples; N_t is the number of test examples (for those cases in which there is a predefined test set). N_C is the number of values of the class attribute; $N_?$ is the number of examples with unknown attribute values.

3 EXPERIMENTAL DESIGN

In this paper we apply XCSL to the five supervised classification tasks summarized in Table 1. All the datasets are taken from the UCI Machine Learning Repository [10]. Three of them (namely `Monks-1`, `Monks-2`, and `Monks-3`), are *synthetic* while `WBC` and `Voting-Record` consist of *real world* data.

For each task, a set *experiments* is performed in which the examples in the dataset are partitioned into a *training set* and a *test set*. The former is used to learn a classification model of the target *concept* while the latter is used to test the predictive accuracy of the classification model previously evolved on unseen cases.

Training Phase. An experiment consists of a number of learning *problems* that XCSL must solve. For each problem a configuration of attribute values is randomly selected from the *training set* and presented to XCSL. The system must *classify* this input configuration, i.e., it must predict the value of the class attribute for the current input configuration. If the prediction is correct XCSL receives a constant reward equal to 1000, zero otherwise. To classify the current input configuration XCSL can *exploit* what it already knows by selecting the most promising class among the available ones. Alternatively, XCSL can *explore* new options by selecting the class randomly. In the former case, we say that XCSL solves the problem *in exploitation* while in the latter we say that XCSL solves the problem *in exploration*. The genetic algorithm is in operation during *exploration* but it does *not* operate during *exploitation*. The covering operator is always enabled, but operates only if needed. At the beginning of a new problem, XCSL decides with probability 0.5 whether it will solve a *learning* problem or a *test* problem. Thus exploration and exploitation problems approximately alternate. During this training phase,

XCSL's performance is computed as the moving average of the reward received in the past 100 exploitation problems.

Testing Phase. When training (i.e., learning) is completed, the solution evolved is tested by predicting the class attribute of the examples in the test set. Each input configuration in the test set is presented only once to XCSL which returns the best prediction (i.e., in *exploitation*). During testing, the genetic algorithm is turned off, as well as the covering operator, and no reward is provided. Final XCSL performance is computed as the percentage of examples in the test set which have been correctly classified.

4 THE MONK'S PROBLEMS

The Monk's problems are a widely used *synthetic* testbed for comparing classification algorithms [2, 14]. The Monk's problems (see Table 1) are defined over six integer attributes (a_1,..., a_6): a_1, a_2, and a_4 have values in {1, 2, 3}; a_3 and a_6 have values in {1, 2}; finally, a_5 has values in {1, 2, 3, 4}. Thus there are 432 distinct configuration of the six variables. There are three Monk's problems. Each problem involves the learning of a target concept described by disjunctions of logical relations among the six variables. In particular, in the first Monk's problem (`Monks-1`) the value of the class attribute (i.e., the target concept) is described by a function which returns: 1 if a_1 is equal to a_2 or the value of a_5 is one; 0 otherwise. [1] In the second Monk's problem (`Monks-2`) the value of the class attribute is: 1 if *exactly* two of attributes are equal to one; zero otherwise. Finally, in third Monk's problem the class attribute is: 1 if $(a_5 = 3 \wedge a_4 = 1) \vee (a_5 \neq 4 \wedge a_2 \neq 3)$; 0 otherwise.

As done in [2], for each problem we trained XCSL on a *training set* containing a *subset* of the possible 432 input configurations (see Table 1) while we tested the final solution on *all* the 432 possible input configurations

The First Problem. We applied XCSL to the first Monk's problem (`Monks-1`) with a population size (N) of 600 classifiers and the following parameter settings: $\beta = 0.1$; $\alpha = 0.1$; $\epsilon_0 = 5$; $\nu = 5$; $\theta_{GA} = 25$; $\chi = 0.8$; $\mu = 0.01$; $\theta_{del} = 40$; $\delta = .1$; condensation starts after 20000 *exploration* problems and last for 10000 problems.[2]

Initially, we applied XCSL on the training set containing 124 labeled examples. The performance of XCSL on the training set, computed as the percentage of correctly classified examples, is depicted in Figure 3 (solid line) with the percentage of macroclassifiers in the population (dashed line). As can be noted, XCSL reaches 100% classification accuracy on the training set. Note that when condensation starts (after 20000 learning problems) predictive accuracy is basically 100% (with rare dips to 99.8%) then during the condensation phase the performance reaches firmly 100% since some inaccurate classifiers have been deleted. Meanwhile, number of macroclassifiers in the population rapidly drops. After 10000 *exploration* problems with condensation turned on the average number of macroclassifiers in the population dropped from 370 macroclassifiers (i.e., 62% of the available population size) to to 14 macroclassifiers (i.e., 2% of the available population size).

Then, we exploited the solutions evolved in the previous experiments to classify the 432 examples in the test set. The results of this step show that XCSL's classification accuracy on the test set is still 100%. From a "crude" performance viewpoint these results confirm those presented in [14] for the basic (ternary) XCS which is higher than that obtained by applying C4.5 [12] on the same train/test sets (see Table 2) or that of other methods (see [2]).

If we look at the final populations evolved we can obtain more insight on the problem. Figure 2 shows one examples of solutions developed by XCSL. It represents the *most compact* solution evolved by XCSL for `Monks-1` which consists of seven macroclassifiers. The four included in the figure represent the complete solution to `Monks-1`; the remaining three (not reported here) had zero prediction and covered the wrong actions cases. In particular, classifier 1 and 2 in Figure 2 cover the two main clauses of the class attribute: classifier 1 covers $a_1 = a_2$ while classifier 2, with condition "$2 > a_5$" covers $a_5 = 1$. Both classifiers 3 and 4 cover the case $(a_1 \neq a_2) \wedge (a_5 > 1)$ which corresponds to all the cases left out by classifiers 1 and 2.

The solution in Figure 2 is *compact* both because the final population has only *seven* macroclassifiers for solving the whole problem *and* because the macroclassifiers have short conditions. But in XCSL classifier conditions are rarely compact [6]. As we noted in [6], XCS has no specific bias toward compact conditions. There-

[1] We found a small discrepancy between the definition of `Monks-1` given with the UCI distribution of the data and the definition given in [2] (used in [14]) which defines `Monks-1` with the predicate: $(a_1 = 1) \wedge (a_5 = 1)$. Here we use the former since it accompanies the data we used in these experiments and it is consistent in the data themselves.

[2] See [1] for an overview of XCS parameters.

id	Condition	Action	P	ϵ	F	N
1	(GT(2)(A5))	1	1000	0	0.96	92
2	(EQ(A2)(A1))	1	1000	0	1.00	121
3	(AND(AND (AND(NOT(EQ(A2)(A1)))(AND(GT(A5)(1)) (NOT(EQ(A2)(A1)))))(NOT(EQ(A2)(A1))))	0	1000	0	0.52	90
4	(AND(AND(AND(NOT(EQ(A2)(A1)))(NOT(EQ(A2)(A1)))) (AND(AND(GT(A5)(1))(4))(NOT(EQ(A2)(A1)))))(NOT(EQ(A2)(A1))))	0	1000	0	0.48	88

Figure 2: The most compact population evolved by XCSL for Monks-1.

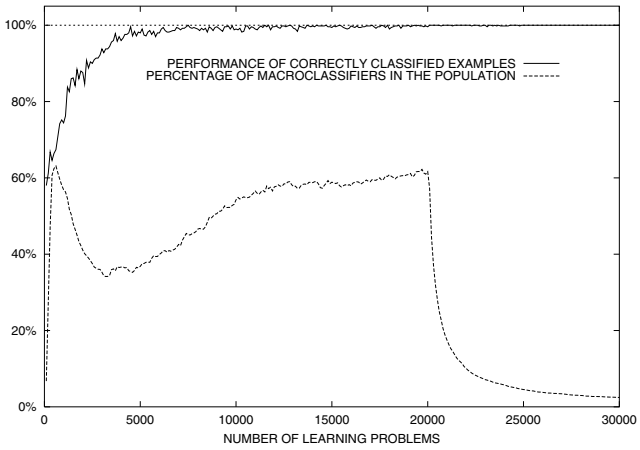

Figure 3: XCSL in Monks-1: Percentage of training examples correctly classified (solid line); Percentage of macroclassifiers in the population (dashed line). Curves are averages over ten runs.

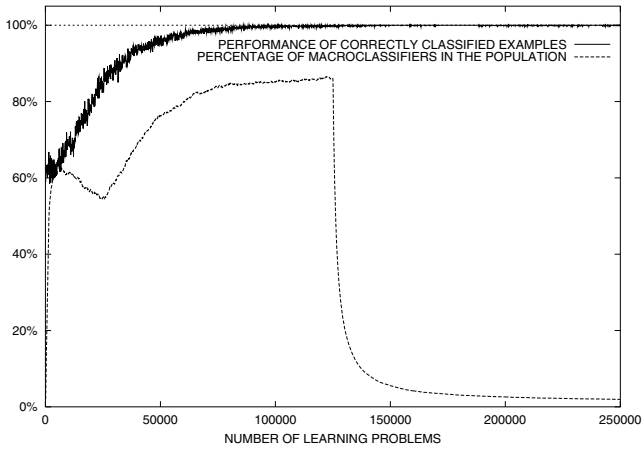

Figure 4: XCSL in Monks-2: Percentage of training examples correctly classified (solid line); Percentage of macroclassifiers in the population (dashed line). Curves are averages over ten runs.

fore, if the representation employed is not bounded in size (as in XCSL) classifier conditions tend to grow as evolution goes on. In particular, we suggest that conditions which exploit ORs are more likely to grow during evolution [6]. Accordingly, compact solutions (like that in Figure 2) do not have OR clauses while other complex solutions XCSL evolved for Monks-1, not be reported here because too complex, exploited OR clauses. However, as we discussed in [6], and briefly at the end of this paper, this "drawback" might be viewed as an interesting feature of the XCS paradigm.

The Second Problem. Monks-2 is more complex than Monks-1 in that it is represented by a complex predicate over the attribute values. As a matter of fact only one particular classification algorithm tested in [2] reached optimal performance.

We applied XCSL to Monks-2 with the same parameter settings used in the previous experiments and a population size (N) of 3000 classifiers; training lasts for 125000 exploration problems, then condensation starts and goes on for other 125000 exploration problems (with crossover and mutation turned off). The classification accuracy of XCSL during training is depicted in Figure 4 (solid line) with the percentage of macroclassifiers in the population (dashed line). We remind the reader that in the population plot "100%" corresponds to the population size (N), i.e., 3000 in this experiment. At the end of this training XCSL has learned to classify almost all the test examples, in fact its predictive accuracy is around 99.8%. Before condensation starts, at 125000, the population contains 2580 classifiers (89% of N); after condensation, at 250000, there are only 59 classifiers on the average (2% of N).

When the solutions evolved are used to classify the test examples, XCSL's classification accuracy drops to 89.6% which is (i) a little bit higher than that reported in [14] for XCS, (ii) higher than that of C4.5 on the same train-test set pair (see Table 2), (iii) but less performing than AQ17DCI [2] which is the only algorithm to reach optimal performance in Monks-2 because of

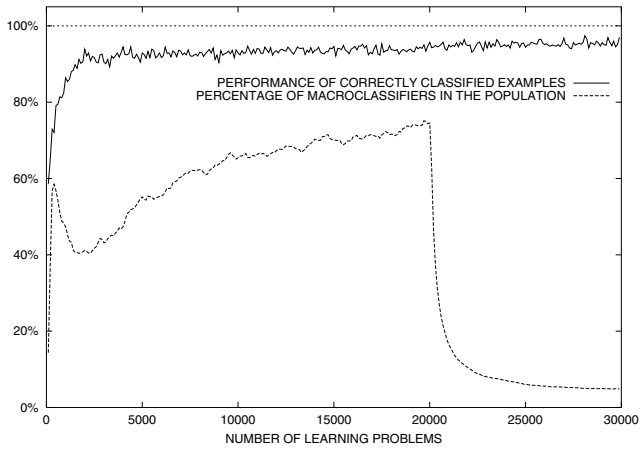

Figure 5: XCSL in `Monks-3`: Percentage of training examples correctly classified (solid line); Percentage of macroclassifiers in the population (dashed line). Curves are averages over ten runs.

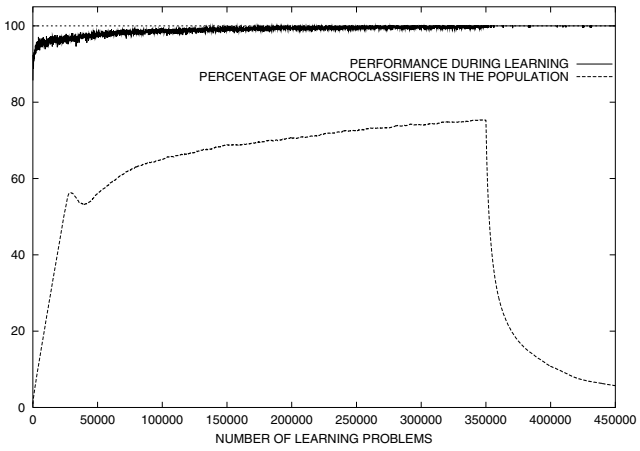

Figure 6: XCSL in `WBC`: Percentage of training examples correctly classified (solid line); Percentage of macroclassifiers in the population (dashed line). Curves are averages over ten runs.

its representation which is capable of representing the concept defined in `Monks-2` [14].

Note that it is not possible to show here any of the solutions developed by XCSL for `Monks-2` since the classifiers evolved for this problems are too complex and would require too much space; we refer the interested reader to [6] for some examples of evolved classifiers.

The Third Problem. `Monks-3` is a little bit more difficult than the previous problems because the training set includes some noise in the form of the six misclassified examples (more or less 5% of the train set). As done before, we applied XCSL on the training set with exactly the same parameters used in `Monks-1`. The classification accuracy of XCSL during training (solid line) and the the percentage of macroclassifiers in the population (dashed line) are depicted in Figure 5. At the end of this phase the predictive accuracy of XCSL is around 95.8%; before condensation starts the average population size is 400 classifiers (70% of N); at the end, the average population size is 24 classifiers (4% of N).

When the solutions evolved is used to classify the test examples, XCSL's classification accuracy is 95.8% which is a little bit less than that of C4.5 for the same train-test pair (see Table 2).

5 THE `WBC` DATASET

The Wisconsin Breast Cancer dataset consists of 699 cases collected by by Dr. William H. Wolberg of the University of Wisconsin Hospitals [9]. It is described by nine attributes which correspond to measured features of tumoral cells; each attribute has values between 1 and 10; there are 16 cases which contain at least an unknown attribute value; there are 458 *benign* cases (i.e., 65.5%) and 241 *malignant* cases (i.e., 34.5%). XCSL has to learn how to discriminate benign cases from malignant ones given the value of nine attributes.

To evaluate the learning capabilities of XCSL on this classification problem we apply a ten-fold crossvalidation procedure [16] which works as follows. Initially, the original set of examples is partitioned in ten *folds*; each fold contains the same distribution of class values; then each fold is taken as a *test set* while the remaining nine folds form a *training set*; XCSL is then applied on the training set for learning a classification model which is then used to classify the examples in the test set. XCSL's performance is computed as the average of classification accuracies obtained on the ten test sets. The parameters of XCSL are set as in the previous experiments except for the population size which is 6400; train lasts for 350000 experiments; condensation for 100000 experiments. At the end of the training phase, the average predictive accuracy of XCSL is 100% while the average population size is 300. We exploited the ten models obtained during training to classify the corresponding test sets created by the cross validation process. The average predictive accuracy of XCSL on the ten test sets is 96.6%.

To compare XCSL with C4.5, we trained C4.5 on the *same* training sets used for XCSL; each model developed by C4.5 was then used to classify the examples

in the corresponding test set. For this purpose, we employed the version of C4.5 available at [13]. The average predictive accuracy of C4.5 on the ten test sets is 94.1% which is lower than the predictive accuracy of XCSL. To test whether the difference between XCSL and C4.5 is statistically significant we apply the *paired* two-tailed t-test and the Wilcoxon matched paired test. These return two p-values equal to 0.01 and 0.005 respectively, i.e., with a confidence level (probability) of 99% XCSL performs *significantly better* than C4.5 on the `WBC` dataset.

6 THE `Voting-Record` DATASET

The `WBC` dataset has only few examples with unknown values. To test XCSL's performance when many unknown attribute values are present, we apply XCSL to the `Voting-Record` dataset [10]. This dataset includes votes for each of the U.S. Congressmen on sixteen key votes. It consists of 435 examples, one for each voter; 267 are labeled as democrats; 168 are labeled as republicans; each example is represented by sixteen attributes, one for each voting; the 5% of the attribute values are unknown. there are two possible attribute values which correspond to the vote. XCSL has to learn how to discriminate between democrats and republicans from the representative voting record. The XCSL's parameters are set as in the previous case except for N that for this problem is set to 3200 classifiers; condensation starts after 200000 exploration problems and lasts for 200000 problems. As done for `WBC`, we use a ten fold crossvalidation and compare the predictive accuracy of XCSL and C4.5 on the ten test sets. XCSL's average classification accuracy on the test sets is 95.7% whereas C4.5 classification accuracy is 96.3%. However, by applying the previous significance tests we find that this small difference in classification accuracy *is not* statistically significant.

7 DISCUSSION

The limited results we presented here, suggest that XCS with symbolic representation might be an interesting approach for extracting useful knowledge from data. From a *predictive* Data Mining viewpoint, XCSL performs in most cases at least as well as traditional methods while it outperforms C4.5 on an important real-world dataset. From a *descriptive* Data Mining viewpoint, XCSL might represent the knowledge extracted from the data in an interesting, readable, form.

On the other hand, we also noted that if the knowledge in the data is complex, the solutions developed might be difficult to analyze and present. As a matter of fact,

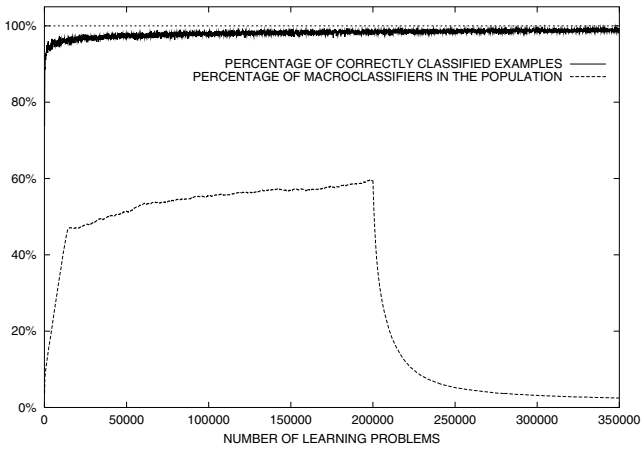

Figure 7: XCSL in `Voting-Record`: Percentage of training examples correctly classified (solid line); Percentage of macroclassifiers in the population (dashed line). Curves are averages over ten runs.

we have not been able to present here some solutions developed by XCSL because they would require many pages of text. Thus the question is *are symbolic representations or general variable length representations interesting*?

Traditional Machine Learning techniques usually have a strong bias toward compact representations, i.e., given the same predictive accuracy in training shorter solutions (i.e., models) are preferred. This criterion, also known as "*Occam's Razor*" is at the base of most classification algorithms. XCS models do not follow this principle and have no bias toward certain kind of hypotheses [6]. Instead, XCS models evolve hypotheses solely on the basis of on-line experience: a rule is better than another if it is more accurate and if it is applies to more cases, i.e., if it is more general (according to Wilson's Generalization Hypothesis [15]). Thus, when applied to variable length (symbolic) representations XCS models tend evolve complex explanations of the phenomenon described in the data.

Although this might appear as a "drawback" in the Data Mining community it has been recently recognized that *Occam's razor* sometimes might represent a limitation for high performing knowledge discovery methods [11] and that discovery/classification methods not biased by *Occam's razor* can perform significantly better than traditional methods, of course, as long as the discovered models are complex but models still comprehensible.

With this respect, XCS models (e.g., [15, 16]), that are not biased by the Occam's razor, appear to be an inter-

Dataset	N	$S_{\texttt{train}}$	$S_{\texttt{cond}}$	[P]	p.a. Training		Testing		
					XCSL	C4.5	XCSL	C4.5	p-value
`Monks-1`	600	20000	10000	14	100.0%	83.9%	100.0%	76.7%	-
`Monks-2`	3000	125000	125000	59	99.8%	65.0%	89.6%	76.3%	-
`Monks-3`	600	20000	10000	24	97.0%	93.4%	95.8%	97.2%	-
`WBC`	6400	350000	100000	300	100.0%	96.0%	96.7%	94.1%	0.01/0.005
`Voting-Record`	3200	200000	150000	78	99.4	97.2%	95.7%	96.3%	0.60/0.444

Table 2: Summary of the results presented in this paper: N is the population size; $S_{\texttt{train}}$ is the number of training experiments; $S_{\texttt{cond}}$ is the number of subsequent training experiments with condensation active; [P] is the average number of macroclassifier in the population at the end of training; "p.a. Training" is the average predictive accuracy reached at the end of training for XCSL and C4.5; "p.a. Testing" is the predictive accuracy on the test set for XCSL and C4.5; "p-value" is the value returned by the significance tests performed for the datasets crossvalidation was used;

esting approach to knowledge discovery applications based on supervised classification. Indeed, variable length representations might become a useful support to such tasks, but techniques to limit the complexity of the evolved solutions must be developed.

References

[1] Martin V. Butz and Stewart W. Wilson. An Algorithmic Description of XCS. Technical Report 2000017, Illinois Genetic Algorithms Laboratory, 2000.

[2] S. B. Thrun et al. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CS-91-197, Carnegie Mellon University, Pittsburgh, PA, 1991.

[3] John H. Holmes. Learning Classifier Systems Applied to Knowledge Discovery in Clinical Research Databases. In Lanzi et al. [8], pages 243–261.

[4] John Koza. *Genetic Programming*. MIT Press, 1992.

[5] Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In Wolfgang Banzhaf et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 345–352. Morgan Kaufmann: San Francisco, CA, 1999.

[6] Pier Luca Lanzi. Generalization in the XCS Classifier Systems with Symbolic Representation. Technical Report 01.??, Dipartimento di Elettronica e Informazione, 2001.

[7] Pier Luca Lanzi and Rick L. Riolo. A Roadmap to the Last Decade of Learning Classifier System Research (from 1989 to 1999). In Lanzi et al. [8], pages 33–62.

[8] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems. From Foundations to Applications*, volume 1813 of *LNAI*. Springer-Verlag, Berlin, 2000.

[9] O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1–18.

[10] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases.

[11] Pedro Domingos. The Role of Occam's Razor in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 4(3), 1999.

[12] R. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kauffmann, Los Altos, California, 1993.

[13] Ross Quinlan. C4.5 Release 8. `http://www.cse.unsw.edu.au/~quinlan/`.

[14] Shaun Saxon and Alwyn Barry. XCS and the Monk's Problems. In Lanzi et al. [8], pages 223–242.

[15] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.

[16] Stewart W. Wilson. Mining Oblique Data with XCS. In *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS-2000), in the Joint Workshops of SAB 2000 and PPSN 2000*, 2000. Extended abstract.

CXCS: Improvements and Corporate Generalization

Andy Tomlinson and Larry Bull
Intelligent Computer Systems Centre
University of the West of England
Bristol BS16 1QY, U.K.
Andy.Tomlinson@uwe.ac.uk Larry.Bull@uwe.ac.uk
phone + 44(0)117 344 3178, +44(0)117 344 3161

Abstract

CXCS applies rule-linkage to Wilson's XCS model. This approach, based on the earlier proposals of Wilson and Goldberg, introduces a macro-level evolutionary operator which creates structural links between rules in XCS and thus forms "corporations" of rules within the classifier system population. CXCS has been shown to offer improved performance over XCS in a series of sequential tasks. In this paper the functionality of CXCS is enhanced to provide increased benefits regarding the same class of tasks and the system's ability to form appropriately generalized solutions is examined.

1 Introduction

Previously our implementation of a corporate classifier system (CCS) (Tomlinson and Bull, 1998) demonstrated that with modification, Wilson and Goldberg's proposals (1987)(see also Smith,1994) regarding rule-clusters in a Michigan-style classifier system (Holland et al., 1986) can offer benefits when applied to a "zeroth-level" classifier system (Wilson, 1994) tackling multiple-step tasks. The system links rules between successive match-sets and employs corporate *persistence* within the performance component. Corporate rules share a common fitness (based on the mean strength of member rules) and the Genetic Algorithm (GA) (Holland, 1975) is enhanced to perform a type of *corporate crossover* operation which produces as offspring, a single hybrid corporation which inherits sections of both parent corporations (see figure 1).

More recently (Tomlinson and Bull, 2000) it was demonstrated that, with some modifications, the linkage mechanisms of CCS can be applied to a system based on XCS (Wilson,1995), and that similar benefits can be achieved regarding the solution of the same series of multiple time-step tasks that CCS was tested in. The resultant system, termed CXCS, evolves corporations whose fitness is assessed according to their consistency in mapping and evaluating certain aspects of these tasks. Some form of internal association within the system is required in order to tackle non-Markov tasks. See also (Tomlinson, 1999) for a comparison of CXCS and an implementation of XCSM (Lanzi, 1998), another accuracy based system that incorporates a memory mechanism. Here CXCS functionality is examined further and mechanisms are presented which enhance system performance in multiple time-step tasks.

2 CXCS

Rules within CXCS are given the same linkage components as those in CCS, i.e. each rule is able to be associated with two other rules by direct reference. These two linkage/reference components are added to the basic genome of rules in CXCS. Initially these linkage components are empty but through the application of a mutation type operator may be set to reference other rules in the system in order to form corporations of rules.

Linkage occurs (again as in CCS) between rules from subsequent match sets at a fixed rate (typically a 10% probability on each step). Rules are not allowed to link between the last time-step of one task and the first step of the subsequent task. On single-step problems corporations are thus not able to form and in such situations CXCS behaves as XCS would. Like the GA, linkage occurs only on exploratory cycles and so is also turned off for the last 1000 trials of testing. In CCS, rule selection for linkage could be either random or probabilistic, or deterministic, based on the relative strengths of rules within the niches. The equivalent parameter to ZCS/CCS rule strength in XCS is the prediction parameter. In XCS it is the accuracy of the prediction that is used to evaluate rules, and it is not in keeping with XCS philosophy to base discovery decisions on the prediction parameter alone. Accuracy and fitness are also discounted as possible weightings for rule selection for linkage. In CXCS it is possible that rules that appear to be inaccurate when evaluated alone are precisely the rules that could benefit from rule-linkage. If the inaccu-

racy is due to some sensory deception then the context of a corporate rule-chain may limit a rule's activation to instances in which its action results in a more predictable consequence. In context the rule becomes more accurate. This is the main motivation for developing CXCS. With this in mind, selection for linkage in CXCS is determined randomly from rules (whose appropriate link is unattached) within the niche, imposing no bias based on the system's current perception of rule utilities.

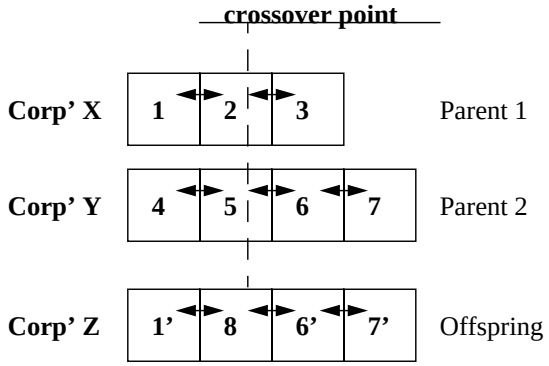

Figure 1: Corporate Crossover

If a corporate rule is selected for deletion then the corporation is first disbanded and then the selected rule is deleted from the rule-base. If a corporate rule is selected for reproduction then the whole corporation is reproduced. The crossover mechanism is expanded to facilitate a form of *corporate crossover* which produces as offspring, a single hybrid corporation which inherits sections of both parent corporations (see figure 1).

As in CCS, corporations are reproduced and evaluated collectively. As such rules within a corporation should share certain parameters used by the discovery component. These are fitness, which determines a rule's chance of selection for reproduction, and the estimate of mean match set size which determines a rule's chance of being selected for replacement; two parameters are introduced, "corporate fitness" and "corporate niche ([M]) size estimate". For single rules these parameters are identical to their existing fitness and match set size estimates. For linked rules, these values can be determined in a number of ways. Each rule could be given the average fitness and match set size estimate of all rules within the corporation. Alternatively, corporate fitness could be based on the lowest exhibited fitness within the corporation. In this way, a corporation is considered only as accurate or fit as its weakest link. This approach certainly offers the theoretical advantage of a bias against unwanted parasites within corporations (Smith, 1994). It is also possible to give each rule in the corporation a corporate niche size estimate equivalent to the smallest represented niche in the corporation. This policy considers that although one rule in a corporation may belong to a well occupied niche or niches, the next rule may be the sole resident in another. In the initial design corporate fitness for each rule in a corporation will be set to the lowest exhibited fitness within the corporation. Corporate niche size estimates will be determined as the mean match set size estimate within that corporate unit.

As in CCS, corporations can, while they continue to match presented stimuli, maintain persistent control of the performance component. So, if on some time-step *t*, a corporate rule takes control of the system, then on the next step, *t+1*, if the next rule in the corporation matches the new stimulus, control is held and the action of this rule automatically becomes the system action at time *t+1*.

In CXCS corporations can take control during both exploration and exploitation cycles, however in this respect functionality differs slightly between the two system modes. On each step, after action selection, during standard performance component cycles a rule is selected from the action set according to some policy and if this rule is corporate (i.e. it has an active link forward) then that corporation is given control of the system. During exploration cycles this rule is selected randomly from the action set [A] and during exploit cycles the rule is selected deterministically according to rule predictions.

Again, as in CCS, *followers* (rules with an active "link-back" component) are given only limited access to the match set [M]. Any rule which has an active "link back" (i.e. a follower) is prevented from entering [M], unless it links back to the rule that is currently in control of the system.

When comparing systems that introduce different numbers of offspring per invocation of the GA it is important to consider the differences in relative rule replacement rates. Without such consideration it is possible to generate quite misleading comparisons of systems as rule replacement concerns tend to be amongst the more fragile aspects of classifier system design (Tomlinson and Bull, 1999b). To counteract this, a variable element is introduced into the CXCS GA activation. The system records the number of rules reproduced on each invocation of the GA (i.e. the size of offspring corporation, S_c). When the existent activation policy indicates that the GA should fire, a further mechanism will only allow the GA to fire with probability set according to the reciprocal of the mean offspring size parameter, S_m (initialized to 1). This estimate is adjusted on each invocation of the GA according to the standard Widrow-Hoff delta rule (Wilson, 1995) with the learning rate parameter β (typically 0.2), i.e. $S_m \leftarrow S_m + \beta(S_c - S_m)$. This modification to the GA activation mechanism ensures at least a more consistent rate of rule replacement throughout

testing, however the drawback is that a corporate system, compared to a standard system will incur a relative reduction in crossover events. The more significant factor is perhaps the rule replacement rate and its effect on convergence within the rule-base, and so here, the variable GA activation policy is adopted for all tests.

3 Minimal Corporate Representation

3.1 Introduction

In any corporate classifier system, due to the possibility of corporations forming and growing arbitrarily throughout the learning process, it is important to ensure that the linkage mechanism operates in a reasonable manner. That is to say, corporations should only survive if they offer some benefit such as, for example, overcoming sensory ambiguities. The enforcement of such regulations on these complex structures is the primary motivation for the use of the anticipation-based corporate fitness approach employed in CXCS. This approach scales corporate fitness according to a corporation's consistency in maintaining control of the performance component, i.e. predicting the environmental outcome of a taken action. The mechanism is based on Wilson's theoretical proposals regarding "expectons" (Wilson, 1995) - see also Anticipatory Classifier Systems (Stolzmann, 1998) for a related mechanism.

If unnecessarily long corporate structures are allowed to proliferate within the population, then successful evolution may be impeded.

3.2 A Test of Minimal Corporate Representation

To ensure that such disruptive eventualities are not a problematic aspect of CXCS design, the system is here tested on a specific, hand-crafted "Delayed Reward Task" (DRT) (Tomlinson and Bull, 1998), the solution of which depends not only on the presence of corporations, but on the systems ability to produce minimally sized corporations, which must overcome non-Markov stimuli but also, must not grow beyond some limited length.

This test of minimal corporate representation in CXCS (see figure 2) consists of two mazes of length four. Only one route through each maze will yield an external reward of 1000, all other routes result in a reward of 0. In previous DRT tests each presented maze is identified on the first time-step of that trial. On subsequent steps, the system receives as stimuli only a "time-code" and so on such steps is unable to determine which of the mazes it is traversing without some form of internal association between successive states. This ensures that the only form of possible inference requires solely internal associations mapping the duration of the trial. XCS having no associative memory is not equipped to tackle such tasks.

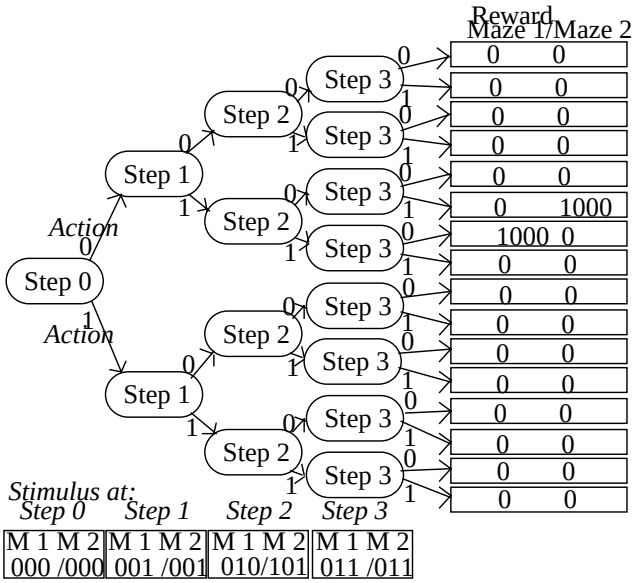

Figure 2: Test for Minimal Corporate Representation

In this test, each maze is identified, not on the first step but on the third step of the trial. As usual, on all other steps the system receives only a time-code. So, on step 0, regardless of which maze the system is presented with, the received stimulus is {0, 0, 0}. On step 1, in either maze the stimulus is {0, 0, 1}. On step 2, in maze 1 the stimulus is {0, 1, 0} but in maze 2 it is {1, 0, 1}, thus differentiating the two mazes. On the final step the stimulus is {0, 1, 1} for both mazes.

In the first maze, the reward winning action sequence is < 0, 1, 1, 0>, in the second maze it is < 0, 1, 0, 1> so in order to successfully predict the consequences of different actions on step 3, some internal association is required (as usual in DRTs). The system always receives the same stimulus on steps 0 and 1, and so may try to form links between these steps. That is acceptable here (although not required) as the consequences of action sequences are consistent in both mazes over these steps (i.e. < 0, 1> is "good", other choices are "bad"). If however the system attempts to link between a "good" single rule firing on step 1 (or a "good" corporation bridging steps 0 and 1) and a "good" corporation that fires on step 2, then the latter firing corporation which previously received a consistent high payoff will now be critically disrupted. The resultant, longer corporation will attempt to fire no matter which maze is being presented, and so inconsistent pay-off will be received. A previously "good" corporation has now become dysfunctional due to inappropriate linkage. To solve this maze optimally, the system is required to produce and maintain appropriate corporations of length two. In this way, this task can be used to test the systems ability to produce min-

imal corporate solutions.

3.3 Results

System parameters for all tests are:

Rulebase Size: P = 800,

Probability of # at an allele position in the initial population: $P_{\#}$ = 0.5,

Initial rule prediction = 10.0,

Learning Rate: β = 0.2,

Discount Factor: γ = 0.71,

Probability of crossover per invocation of the GA: χ = 0.8,

Probability of mutation per allele in the offspring: μ = 0.01,

If the prediction of [M] is less than ϕ times the pop' mean, covering occurs: ϕ = 0.5.

GA activation threshold parameter = 25,

Number of single-rules or corporations produced by GA as offspring per invocation, 1.

Initial rule error= 0.0

Initial rule fitness = 10.0

accuracy function parameter: e_0 = 0.01

accuracy function parameter:α = 0.1

Linkage Rate = 0.1,

Macro-classifiers (Wilson, 1995) are not incorporated.

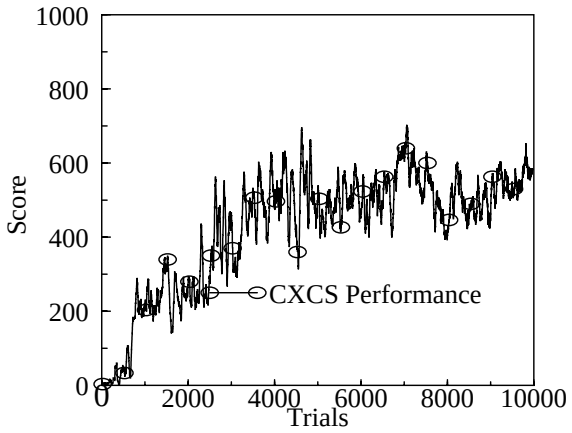

Figure 3: Performance in Minimal Corporate Representation test

Performance plots here represent the average reward in the last 50 exploit problems, and all curves are averages of ten runs. Results (see figure 3) suggest that CXCS can solve this task but also that it has some difficulty in doing so. At the end of testing the system has achieved an average success rate of about 58%. The result is reasonably consistent with results in DRT 2:4 i.e. a standard 2-maze DRT of length 4 time-steps (Tomlinson, 1999), and yet considering that the solution here relies on the formation of only length two corporations performance levels should be somewhat higher. This suggests that although CXCS has not completely failed on this task it could perhaps benefit from further encouragement to regulate linkage activity. The next section investigates the incorporation of link inhibitors to CXCS and presents a performance plot of the modified system in this same task.

4 Link Inhibitors

4.1 Adding Link Inhibition

Link inhibitors have previously been shown to offer performance benefits to CCS in certain multiple time-step tasks (Tomlinson and Bull, 1999a). Here, they are incorporated into CXCS and shown to offer similar benefits. Rules in CXCS have two corporate links, each of which, when active, will reference another rule in the system. This is the nature of corporate rule structures. The introduction of link inhibitors allows rules to evolve that are unable to join to other rules. On rule creation, either on formation of the population at the beginning of testing, or due to the covering mechanism there is a fixed probability that one or both of its links will be inhibited. Each link is considered in turn. A probability of 1 in 4 (as for CCS) of rule links are inhibited. Again, as in CCS, and like other rule attributes, link inhibitors are inherited from parents to offspring. If crossover is employed, then the offspring inherits the "inhibit back" of the first parent and the "inhibit forward" of the second, in a similar manner to the links themselves. In tasks such as the "minimal corporate representation task" (above) it is anticipated that the system will benefit from the capability to evolve such restricted concepts.

4.2 Results

With link inhibition included, CXCS is now tested on the above described minimal corporate representation task. Parameters are unaltered from the previous test settings.

Performance has now improved to about 78% at the end of testing (figure 4). This is a significant improvement over the previous result and more in line with expectations. The solution of this task relies on the formation of corporations of length two in order to map the two mazes. Examination of the rulebase reveals predominantly length two corporations. Below are some typical examples of corporations that lead the system along the "high payoff" routes through the mazes:

Corporation 2179 successfully navigates the latter half of the first maze. There are many examples of this complex concept in the rulebase.

Note: <o> indicates an active link inhibitor, - simply indi-

cates an inactive link.

ID	corp ID	cond	Action	link <-	link ->	Pred	Err	Fit
8001	2179	# 1 0	1	<o>	8002	710	0	1
8002	2179	# # #	0	8001	<o>	1000	0	1

Corporation 6383 successfully navigates the latter half of the second maze. Again, there are many examples of this complex concept present at the end of testing.

ID	corp ID	cond	Action	link <-	link ->	Pred	Err	Fit
8016	6383	1 0 #	0	<o>	8017	710	0	1
8017	6383	0 # #	1	8016	-	1000	0	1

Corporation 1983 “successfully” matches the first half of both mazes, however it does exhibit significant error levels.

ID	corp ID	cond	Action	link <-	link ->	Pred	Err	Fit
6932	1983	# 0 0	0	<o>	6933	231	82	0.77
6933	1983	# # #	1	6932	<o>	337	396	1

It is perhaps worth mentioning at this point that rule 6933 receives a high fitness as when it is active, being a “corporate follower”, it has exclusive access to the action set, and so its relative accuracy will tend to be 1, resulting in the optimal fitness value. Corporate fitness however (the parameter considered by the GA) will be 0.773 (i.e. the lowest fitness within the corporation (Tomlinson and Bull, 2000)).

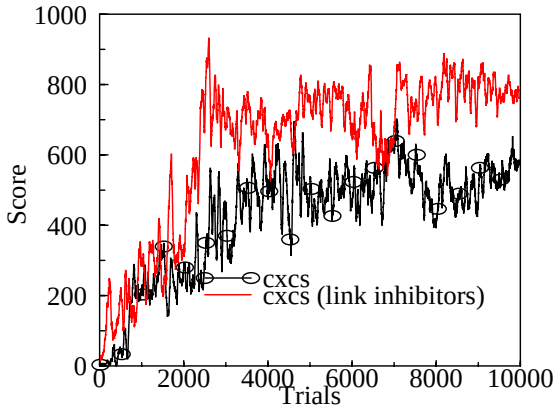

Figure 4: Performance comparison in minimal corporate representation test

It is also worth mentioning that the resultant rulebase, at the end of testing, contains many accurate corporate concepts that lead to 0 payoff (i.e. predict that wrong routes through the mazes lead to no reward). In fact both mazes are fully mapped in this respect. Such concepts also have reasonable numerosities. However, in such tasks, concern must focus on the system’s abilities to actually find the good solutions/routes.

Considering individual runs the basic system achieved optimal performance in three runs, the system with link inhibitors reached optimal performance in four runs. CXCS is now further enhanced by the inclusion of another mechanism.

5 Direct Corporate Payoff

5.1 Introduction

Previously Direct Corporate Payoff has been shown to offer benefits to CCS in multiple time-step tasks by providing accelerated reinforcement to the earlier firing members of corporate rule structures (Tomlinson and Bull, 1999a). Here, the mechanism, which modifies the functionality of the performance component, is incorporated into CXCS. Due to the nature of XCS, the implementation is necessarily slightly different to that of the CCS version. These modifications are discussed and then results are presented which illustrate that similar benefits to those gained in CCS can also be obtained in CXCS.

5.2 Implementing Direct Corporate Payoff in CXCS

Once a corporation has taken control of the performance component on some time step, on all subsequent time steps, while control is maintained, the action set will consist solely of the currently active member of that corporation. This “persistence” (Tomlinson and Bull, 1998) is necessary to encapsulate the corporation regarding its evaluation and thus facilitate the previously demonstrated benefits. This is true both for CCS and CXCS. So, once the first rule in the corporation gains control, then it is guaranteed that, at least under normal circumstances, the subsequent member rules will also fire.

At some time, the last firing rule will receive some reward from the environment or alternatively some internal payoff from the subsequent action set. When direct corporate payoff is employed, this rule adjusts its utility in the standard system manner, but rather than apply the learning rate to internal corporate pay-off, the rule passes back a discounted version of its own utility which becomes the utility of the previous rule in the corporation. This process contin-

ues until the utility of the first firing rule has been overwritten. Actually in CCS, this payoff mechanism is applied on each subsequent step of corporate control, and rule utilities may be overwritten several times during this period.

In CXCS, which employs the more involved XCS rule evaluations, it is necessary to be a little more cautious when considering this process. In XCS, a rule's usefulness is depicted by three parameters, prediction, error and accuracy. So, the first issue is to determine if, in addition to the payoff value, any system parameter(s) should be passed from one rule to its predecessor in the above described manner during the direct payoff operation.

If payoff alone is passed back, then prediction will still be gradually adjusted according to the XCS reinforcement process (the delta rule). This will not result in accelerated evaluation of rule utilities, so it was decided that rule predictions will be passed back along with the payoff values. Both prediction and payoff are discounted and these values become the prediction and payoff of the previous firing rule.

The accuracy of a corporate rule in CXCS is determined not only by its consistency of payoff, but also by the consistency of its corporate link to correctly predict some anticipated stimulus (i.e. a rule's consistency in maintaining control, once it has gained it). It seems unwise for a rule to have some version of this parameter passed back by the subsequent firing rule in the corporation.

Error is determined as the difference between a rule's expected payoff (or prediction) and its actual payoff. As a discounted prediction is passed back, along with the discounted payoff, the rules error parameter can reliably be calculated as usual for XCS and does not need to be passed back.

So prediction and payoff are passed back and then rule status is evaluated in the usual manner. Although this process involves updating the prediction value according to the delta rule, during periods of direct payoff (i.e. control periods), for each rule these parameters both adjust at a rate consistent with the subsequent firing rule in the corporation and so benefits of accelerated reinforcement can be observed.

Finally, if direct payoff occurs, as in CCS, on each time step and subsequent payoff operations simply overwrite previous values during control periods then further complications arise. The benefits of direct payoff to a corporation occur on the early evaluations after its creation, when the accelerated payoff avoids the reinforcement delays associated with purely local schemes. However on the first periods of corporate control, if prediction and payoff values are overwritten for each subsequent firing rule on several time steps during the control period then these rules will exhibit weak accuracy values due to their continual parameter adjustments (error being adjusted according to the delta rule as usual). For this reason, there is no parameter adjustment at all until control is lost, either due to failure to match some stimulus or natural control resolution or the receipt of some reward from the environment. At this point direct payoff is activated. Clearly if control ends for some reason other than the receipt of external reward, then the operation must occur in $[A]_{-1}$ (i.e. on the subsequent time step).

With direct payoff incorporated into CXCS and the preceding modifications implemented, the system is again tested in a series of standard DRTs.

5.3 Results

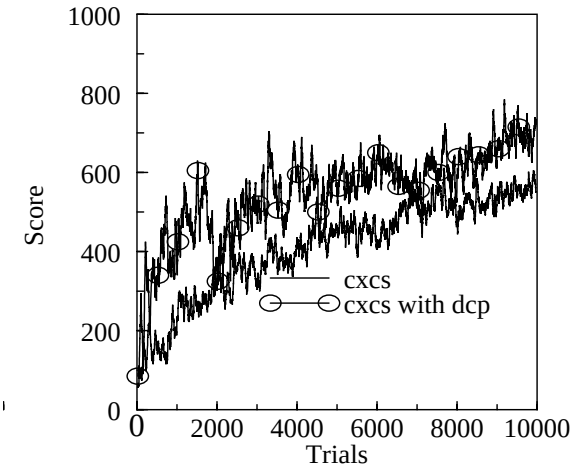

Figure 5: Performance in DRT 4:3

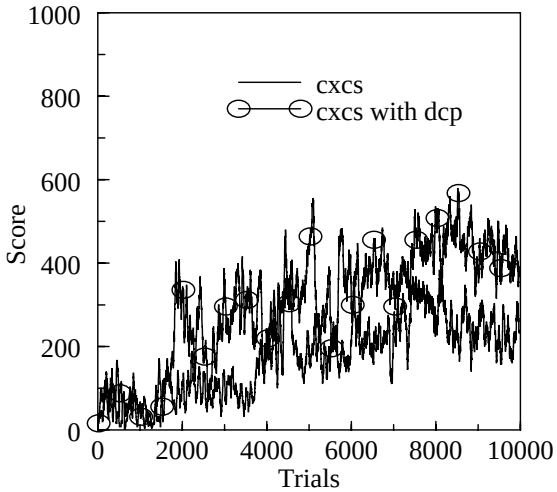

Figure 6: Performance in DRT 4:4

With direct payoff incorporated as described in the previous section, CXCS is tested in two standard delayed reward tasks. Both are four-maze tasks; one of length three and the other of length four. Results (figures 5 and 6) indicate performance improvements over those of the system without the direct payoff mechanism, in a similar manner to the improvements observed when the mechanism was applied to CCS. In task 4:3 four runs exceed 75% performance, and in task 4:4 five runs exceed 50%. As would be expected, increased improvements can be seen as the task length increases.

6 Corporate Generalization in CXCS

6.1 Introduction

Accuracy was introduced into XCS to facilitate improved rule evaluations and to provide a means of discrimination between appropriately general and over general rules. Anticipation-based corporate fitness was introduced into CXCS with similar motivations. In this section, a brief investigation into CXCS's ability to produce accurately general corporate rule structures is presented. To facilitate this analysis, a specific, hand-crafted DRT is first introduced.

6.2 A Test of Corporate Generalization

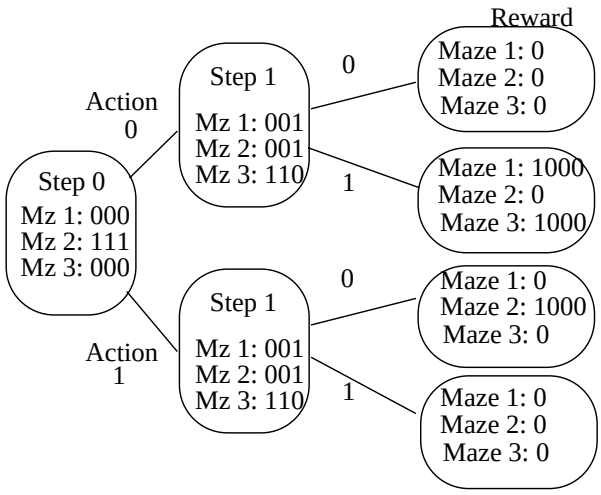

Figure 7: Simple test for corporate generalization

A test is now presented which illustrates, in a somewhat simplistic manner, CXCS's capabilities to form generalized corporate concepts (see figure 7). The test is not a difficult one for the system, however analysis of the rulebase after testing will reveal whether the system has managed to form appropriate generalizations or if it has simply derived a series of unnecessarily specialized solutions. The test is a DRT which consists of three mazes of length two. As usual, only one route through each maze will yield a reward of 1000, the other three resulting in no external reward.

Mazes one and three are identical in every way except that the stimuli on the second time-step differ. So the correct route through each involves the same sequence of actions. The other maze (maze two) is included simply to ensure that corporations are necessary for the solution of this task. It has a unique stimulus on time-step one and the same stimulus on the second time-step as the first maze. The required action for this maze on step two however is different.

6.3 Results

As anticipated, the system has no difficulty in solving this task. A plot of performance is presented below (figure 8).

The rulebase (size 400) was examined after testing, and consisted predominantly of corporations of size 2. Typically after testing the rule-base contains about 170 corporations all of size 2. This accounts for about 340 of the 400 rules. These corporations fully mapped the mazes and consistently exhibited appropriate prediction and fitness values. In all ten runs the system reached optimal performance.

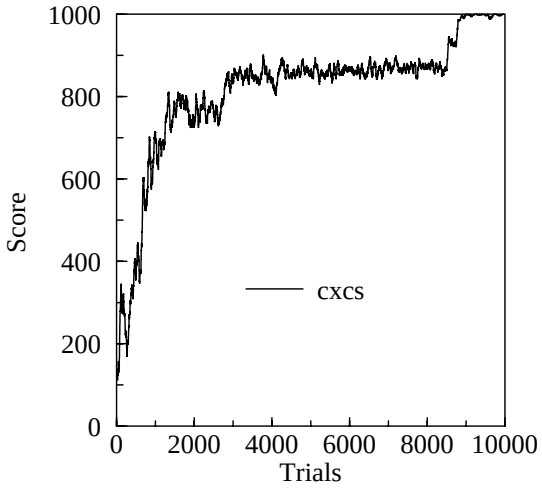

Figure 8: Performance in corporate generalization test

Regarding corporations that portrayed successful routes

through the mazes, it is clear that the system evolved generalized concepts. Example rules are included below:

Corporation 9436 navigates mazes one and three successfully and exhibits optimal generalization. This is an example of the type of corporation that this test was designed to illustrate. Specifically rule 3247 is fully general as required to yield a reward of 1000 from both mazes (see figure 7).

ID	corp ID	cond	Acti on	link <-	link ->	Pred	Err	Fit
3273	9436	0 # 0	0	<o>	3274	709	0	0.99
3274	9436	# # #	1	3273	-	999	0	1

Corporation 9427 navigates maze two successfully and also exhibits optimal generalization.

ID	corp ID	cond	Acti on	link <-	link ->	Pred	Err	Fit
3265	9427	# 1 1	1	-	3266	709	0	0.99
3266	9427	# # #	0	3265	-	999	0	1

Typically many copies of these corporate concepts were present in the rulebase (especially 9436 which mapped two of the three mazes). Corporations which depicted unsuccessful routes through mazes also exhibited reasonable generalizations, as would be expected (not shown).

7 Conclusions

This work has considered enhancements to the CXCS design and has analysed the system regarding certain important characteristics. It has been tested on its ability to produce minimal corporate solutions and the inclusion of link inhibitors has been shown to offer benefits here. Direct corporate payoff was then added and the accelerated reinforcement mechanism was again shown to improve performance. Finally CXCS was tested on its ability to produce optimally generalized corporate solutions. System characteristics here were found to conform to expectations.

8 References

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Holland, J. H., Holyoak, K. J., Nisbett, R. E. & Thagard, P.R. (1986) *Induction: Processes of Inference, Learning and Discovery.* MIT Press.

Lanzi, P. L. (1998) “An Analysis of the Memory Mechanism of XCSM”, In Eiben, A.E., Back, T., Schoenauer, M. & Schwefel, H.P.(Eds.) *The Fifth International Conference on Parallel Problem Solving from Nature* (pp 501-510), Springer.

Smith, R. E. (1994) “Memory exploitation in learning classifier systems.” *Evolutionary Computation*, 2(3): 199-220.

Stolzmann, W. (1998) “Anticipatory Classifier Systems.“In Koza, J.R. et al. (Eds.) *Genetic Programming 1998. Proceedings of the Third Annual Conference* (pp.658-664), Morgan Kaufmann.

Tomlinson, A. (1999) “Corporate Classifier Systems”, Ph.D. Thesis, Faculty of Computer Studies and Mathematics, University of the West of England.

Tomlinson, A. & Bull, L (1998) “A Corporate Classifier System.” In Eiben, A.E., Back, T., Schoenauer, M. & Schwefel, H.P.(Eds.) *The Fifth International Conference on Parallel Problem Solving from Nature* (pp 550-559), Springer.

Tomlinson, A. and Bull, L. (1999a) “On Corporate Classifier Systems: Increasing the Benefits of Rule-linkage” In Banzhaf, W., Daida, J., Eiben, A.E.,Garzon, M.H.,Honavar, V.,Jakiela, M., Smith, R.E. (Eds.) *Proceedings of the 1999 Genetic and Evolutionary Computation Conference* (pp 649-656), Morgan Kaufmann.

Tomlinson, A. and Bull, L. (1999b) “A Zeroth Level Corporate Classifier System.” In Wu, A.S. (Ed.) *Proceedings of 1999 Genetic and Evolutionary Computation Conference Workshop Program* (pp 306 - 313), Morgan Kaufmann.

Tomlinson, A. and Bull, L. (2000) “A Corporate XCS.” In Lanzi, P. L., Stolzmann, W. & Wilson, S. W. (Eds.) *Learning Classifier Systems: From Foundations to Applications (Lecture notes in computer science 1813)* (pp 195 - 208), Springer.

Wilson, S. W. & Goldberg, D. E. (1989) “A critical review of classifier systems.” In Schaffer, J. D. (Ed.) *Proceedings of the Third International Conference on Genetic Algorithms*, (pp.244-255), Morgan Kaufmann.

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): 149-176.

Function Approximation with a Classifier System

Stewart W. Wilson
Prediction Dynamics, Concord MA 01742
Department of General Engineering
The University of Illinois at Urbana-Champaign IL 61801
wilson@prediction-dynamics.com

Abstract

A classifier system, XCSF, is introduced in which the prediction estimation mechanism is used to learn approximations to functions. The addition of weight vectors to the classifiers allows piecewise-linear approximation. Results on functions of up to six dimensions show high accuracy. An interesting generalization of classifier structure is suggested.

1 Introduction

Classifier systems estimate payoff. That is, the rules (classifiers) evolved by a classifier system each keep a statistical estimate of the payoff (reward, reinforcement) expected if the classifier's condition is satisfied and its action is executed by the system. In XCS, a particular classifier system architecture (Wilson 1995), the classifiers form quite accurate payoff estimates, or *predictions*, since the classifiers' fitnesses under the genetic algorithm depend on their prediction accuracies. In effect, XCS approximates the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$, where X is the set of possible inputs, A the system's set of available actions (typically finite and discrete), and P is the set of possible payoffs. If attention is restricted to a single action $a_i \in \mathrm{A}$, the mapping has the form $\mathrm{X} \times a_i \Rightarrow \mathrm{P}$, which is a function from input vectors x to scalar payoffs. Thus the system approximates a separate function for each a_i.

In the reinforcement learning contexts in which classifier systems are typically used, the reason for forming these payoff function approximations is to permit the system to choose, for each x, the best (highest-paying) action from A. However, there are contexts where the output desired from a learning system is not a discrete action but a continuous quantity. For instance in predicting continuous time series, the output might be a future series value. In a control context, the output might be a vector of continuous quantities such as angles or thrusts. Apart from classifier systems based on fuzzy logic (Valenzuela-Rendón 1991; Bonarini 2000), there are none which produce real-valued outputs. Our hypothesis was that the payoff function approximation ability of XCS could be adapted to produce real-valued outputs, as well as be used for function approximation in general applications. Our objective in this paper is to examine, as a first step, XCS's potential for learning approximations to simple functions.

In the paper we adapt XCS to learn functions of the form $y = f(x)$, where y is real and x is a vector with integer components $x_1, ...x_n$. Our results demonstrate approximation to high accuracy, together with evolution of classifiers that tend to distribute themselves efficiently over the input domain. As a by-product of the research, a conceptual generalization of classifier structure is developed.

The next section describes modifications of XCS for function approximation. Section 3 has results on a simple piecewise-constant approximation. In Section 4 we introduce a new classifier structure that permits piecewise-linear approximations. Results on simple functions are shown in Section 5. In Section 6 we demonstrate accurate approximation of a six-dimensional function. Section 7 presents the classifier structure generalization. The final section has our conclusions and suggestions for future work.

2 Modification of XCS

XCS was modified in two respects (later, a third). The first was to adapt the program for integer instead of binary input vectors. The second, very simple, was to make the program's payoff predictions directly accessible at the output and to restrict the system to a single (dummy) action. The resulting program was called

XCSF. (We omit a description of basic XCS, but refer the reader to Wilson (1995), Wilson (1998), and the updated formal description in Butz and Wilson (2001) that XCSF follows most closely.)

The changes to XCS for integer inputs were as follows (drawn from Wilson (2001)). The classifier condition was changed from a string from {0,1,#} to a concatenation of "interval predicates", $int_i = (l_i, u_i)$, where l_i ("lower") and u_i ("upper") are integers. A classifier matches an input x with attributes x_i if and only if $l_i \leq x_i \leq u_i$ for all x_i.

Crossover (two-point) in XCSF operates in direct analogy to crossover in XCS. A crossover point can occur between any two alleles, i.e., within an interval predicate or between predicates, and also at the ends of the condition (the action is not involved in crossover). Mutation, however, is different. The best method appears to be to mutate an allele by adding an amount $\pm rand(m_0)$, where m_0 is a fixed integer, *rand* picks an integer uniform randomly from $(0, m_0]$, and the sign is chosen uniform randomly. If a new value of l_i is less than the minimum possible input value, in the present case 0, the new value is set to 0. If the new value is greater than u_i, it is set equal to u_i. A corresponding rule holds for mutations of u_i.

The condition of a "covering" classifier (a classifier formed when no existing classifier matches an input) has components $l_0, u_0, ..., l_n, u_n$, where each $l_i = x_i - rand_1(r_0)$, but limited by the minimum possible input value, and each $u_i = x_i + rand_1(r_0)$, limited by the maximum possible input value; $rand_1$ picks a random integer from $[0, r_0]$, with r_0 a fixed integer.

For the subsumption deletion operations, we defined subsumption of one classifier by another to occur if every interval predicate in the first classifier's condition subsumes the corresponding predicate in the second classifier's condition. An interval predicate subsumes another one if its l_i is less than or equal to that of the other and its u_i is greater than or equal to that of the other. For purposes of action-set subsumption, a classifier is more general than another if its *generality* is greater. Generality is defined as the sum of the widths $u_i - l_i + 1$ of the interval predicates, all divided by the maximum possible value of this sum.

3 Piecewise-Constant Approximation

The simplest way to approximate the function $y = f(x)$ with XCSF is to let x be the input and y the payoff. After sufficient sampling of the input space, the system should, given an x, more or less accurately predict the corresponding y. In all XCS-like systems the fitness of a classifier depends on its accuracy of prediction, so that XCSF should converge to a population of classifiers that, over their respective input ranges, predict payoff well. The closeness of the approximation should be controllable with the *error threshold* ϵ_0, as follows. A classifier with *prediction error* ϵ_1 has higher *accuracy* than a classifier with error ϵ_2 if $\epsilon_1 < \epsilon_2$ (Butz and Wilson 2001). Since classifier *fitness* depends on accuracy, classifiers with lower errors will win out in the evolutionary competition. However, by definition, classifiers with errors *less* than ϵ_0 have constant fitness, so no further fitness pressure applies. Thus ϵ_0 should limit the closeness of the approximation.

It is important that besides evolving accurate classifiers, the system employ the classifiers efficiently over the input domain. In slowly-changing or low-gradient regions of the function we would hope to evolve classifiers with relatively large interval predicates. The function value, and thus a given classifier's error, would change relatively little over such regions; so, as in XCS, a tendency toward accurate, maximally general conditions (Wilson 1995) should cause the interval predicates to expand. Conversely, we should expect classifiers with small interval predicates where the function is changing rapidly. We also hope for an efficient *distribution* of classifiers over the domain in the sense that the tiling minimizes overlaps between classifier predicates.

Figure 1 shows typical results from an experiment in which XCSF approximated the function $y = x^2$, a parabola, over the interval $0 \leq x < 100$. (In all experiments reported here, the input value range was [0,99].) Plotted are XCSF's prediction for each possible x as well as the function itself.

The system learned from a dataset consisting of 1000 x, y pairs, with x chosen randomly and y the corresponding function value. In the experiment, a pair was drawn randomly from the dataset, its x value presented to XCSF as input, and the y value used as reward or reinforcement. XCSF formed a match set [M] of classifiers matching x and calculated the *system prediction* for each possible action in the usual way; since there was only one, dummy, action, just one system prediction was calculated and that became the system's output. An *action set* [A] was formed consisting of classifiers in [M] having the dummy action (i.e., all classifiers in [M]), and the predictions of the classifiers in [A] were adjusted using the reward, y, in the usual way; the other classifier parameters were also adjusted. A genetic algorithm was run in [A] if called for. This cycle was repeated 50,000 times after which the plot in Figure 1 was obtained by sweeping through all pos-

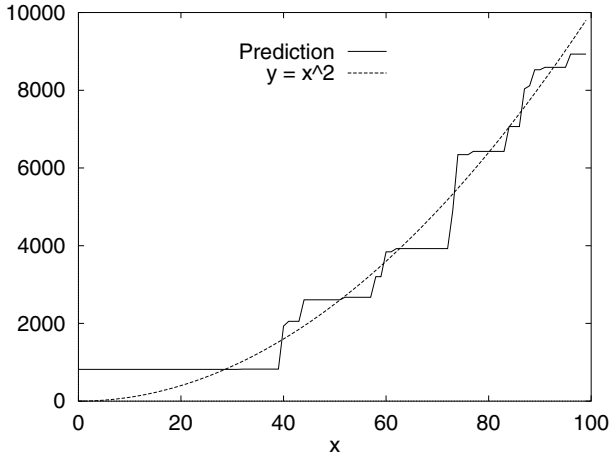

Figure 1: Piecewise-constant approximation to the parabola function $y = x^2$. $\epsilon_0 = 500$.

	CONDITION	PRED	ERR	FITN	NUM
0.	|.....oO|	8992.	283.	.796	31
1.	|.....Oo.|	7876.	322.	.906	32
2.	|.....oOO...|	6162.	503.	.564	20
3.	|.....OOO...|	5990.	570.	.188	9
4.	|.....oOOO...|	5988.	569.	.028	1
5.	|.....oOOo...|	5668.	413.	.075	3
6.	|.....oOO....|	5534.	281.	.090	3
7.	|.....oOo....|	5423.	205.	.298	10
8.	|.....oOOOo...|	5053.	524.	.002	1
9.	|.....oOOo....|	4861.	465.	.054	5
10.	|.....OOOo.....|	3854.	468.	.862	30
11.	|.....oOOOO.....|	1481.	364.	.135	7
12.	|.....OOOOO.....|	1359.	377.	.358	18
13.	|OOOOOOOOo.....|	821.	378.	.372	15
14.	|OOOOOOOo.....|	820.	377.	.371	15

Figure 2: Classifiers from experiment of Figure 1. (PREDiction, ERRor, FITNess, NUMerosity.)

sible values of x and recording the resulting system predictions.

Parameter settings for the experiment were as follows, using the notation of Butz and Wilson (2001): population size $N = 200$, learning rate $\beta = 0.2$, error threshold $\epsilon_0 = 500$, fitness power $\nu = 5$, GA threshold $\theta_{GA} = 48$, crossover probability $\chi = 0.8$, mutation probability $\mu = 0.04$, deletion threshold $\theta_{del} = 50$, fitness fraction for accelerated deletion $\delta = 0.1$. In addition: mutation increment $m_0 = 20$ and covering interval $r_0 = 10$. GA subsumption was enabled, with time threshold $\theta_{GAsub} = 50$.

Several aspects of Figure 1 are of interest. The Prediction curve has a "staircase" appearance typical of a piecewise-constant approximation. The height of the major "steps" varies between about 1000 and 2000. Examination of individual steps indicates an average error roughly consistent with the value of ϵ_0, suggesting that ϵ_0 is controlling the closeness of the approximation. The width of the steps is, again roughly, wider in the "flatter" part of the function and narrower in the steep part. Finally, it is significant that the Prediction curve indeed takes the form of a staircase, instead of being smoother. Long flat "steps" suggest that one set of classifiers is in control (forms [M]) after which, on the next step, another set takes over. This in turn suggests a tendency toward efficient distribution of classifier resources over the domain.

Figure 2 gives another perspective. It is a listing of the (macro)classifiers of the population at the end of the experiment, 15 in all. Shown are each classifier's condition, prediction, error, fitness, and numerosity. Note that most classifiers with substantial fitnesses have errors less than ϵ_0. A special graphic notation is used to represent the condition. Since x has just one component, the condition contains just one interval predicate. The possible range of x, 0-99, is divided into 20 equal subranges. An interval predicate is indicated by a cluster of "O"s that covers its range. If an interval predicate entirely covers a subrange, e.g., 35-39, an "O" is placed at that range's position. If the interval predicate covers some of but less than the whole of a subrange, a small "o" is put there. This notation has been found more perspicuous than using the raw numbers.

Note how, consistent with Figure 1, the classifier conditions are larger toward the beginning of the domain, where the function slope is lower. It is also interesting that the classifiers with higher fitnesses and numerosities cover the domain without a great deal of overlap. These classifiers dominate the calculation of the system prediction, since the latter is a fitness-weighted average of the predictions of matching classifiers. Because they dominate, the Prediction curve takes the form of a staircase. Apart from the presence of the remaining, lower fitness, classifiers, the distribution of resources over the domain is thus relatively efficient.

In sum, XCSF succeeds in approximating the function in accordance with a stated error criterion (confirmed for additional values of ϵ_0) and the classifiers are employed reasonably well. Still, a piecewise-constant approximation is primitive compared with an approximation where the approximating segments more closely follow the function's contour. The simplest such approach is a *piecewise-linear* approximation. But how could a piecewise-linear approximation be done with a classifier system?

4 Piecewise-linear Approximation

Traditionally, a classifier's prediction is a number intended to apply for all inputs x that satisfy its condition. However, for function approximation, it would be desirable if the prediction could vary over the condition's domain, since the function being approximated generally varies. In effect, the prediction itself should be a function, the simplest form of which would be a linear polynomial in the input components, call it $h(x)$. The function $h(x)$ would substitute for the classifier's traditional (scalar) prediction, p. Then, given an input x, each matching classifier would *calculate* its prediction by computing $h(x)$.

For approximating a one-dimensional function $f(x)$, $h(x)$ would be a two-term polynomial $h(x) = w_0 + w_1 x_1$. In this case, w_1 can be thought of as the slope of an approximating straight line, with w_0 its intercept. For an n-dimensional $f(x)$, $h(x) = w \cdot x'$, where w is a *weight vector* $(w_0, w_1, ..., w_n)$ and x' is the input vector x augmented by a constant x_0, i.e., $x' = (x_0, x_1, ..., x_n)$. In this case $h(x)$ computes a *hyperplane approximation* to $f(x)$. Classifiers would have different weight vectors w since in general the domains of their conditions differ.

Of course, the classifiers' weight vectors must be adapted. If classifiers are to predict with a given accuracy, the coefficients w_i of their weight vectors must be appropriate. One approach is use an evolutionary algorithm. The weight vector would be evolved along with the classifier condition. For this, the w_i could be concatenated with the interval predicates of the condition and the whole thing evolved as a unit. Or, it might be preferable to use separate processes: there is reason to think the *Evolutionsstrategie* might be more suitable than the GA for the weight vector. In the present work, however, we did not use an evolutionary technique for the weight vector, but instead adapted it using a modification of the *delta rule* (Mitchell 1997).

The delta rule is given by

$$\Delta w_i = \eta(t - o)x_i,$$

where w_i and x_i are the ith components of w and x', respectively. In the quantity $(t - o)$, o is the output, in the present case the classifier prediction, and t is the *target*, in this case the correct value of y according to $y = f(x)$. Thus $(t - o)$ is the amount by which the prediction should be corrected (the negative of the classifier's instantaneous error). Finally, η is the *correction rate*. The delta rule says to change the weight proportionally to the product of the input value and the correction.

Notice that correcting the w_i in effect changes the output by

$$\Delta o = \Delta w \cdot x' = \eta(t - o)|x'|^2.$$

Because $|x'|^2$ is factored in, it is difficult to choose η so as to get a well-controlled overall rate of correction: η too large results in the weights fluctuating and not converging; if η is too small the convergence is unnecessarily slow. After some experimentation with this issue, we noticed that in its original use (Widrow and Hoff 1988), the correction rate was selected so that the entire error was corrected in one step; this was possible, however, because the input vector was binary, so its absolute value was a constant. In our problem, reliable one-step correction would be possible if a *modified delta rule* were employed:

$$\Delta w_i = (\eta/|x'|^2)(t - o)x_i.$$

Now the total correction would be strictly proportional to $(t - o)$ and could be reliably controlled by η. For instance, $\eta = 1.0$ would give the one-step correction of Widrow and Hoff. In the experiments that follow, we used the modified delta rule with various values of $\eta \leq 1.0$.

Use of a delta rule requires selection of an appropriate value for x_0, the constant that augments the input vector. In tests, we found that if x_0 was too small, weight vectors would not learn the right slope, and would tend to point toward the origin. Choosing $x_0 = 100$ solved this problem, perhaps because it was then of the same order of magnitude as the other x_i.

For piecewise-linear approximation, no changes were necessary to XCSF except for addition of the weight vectors to the classifiers, and provision for calculation of the predictions and application of the modified delta rule to the action set classifiers on every time-step. In a classifier created by covering, the weight vector was randomly initialized with weights from [-1.0,1.0]; GA offspring classifiers inherited the parents' weight vectors. Both policies yielded performance improvements over other initializations. In the experiments, most parameter settings were the same as those given in Section 3; differences will be noted. Settings of the new parameter, η, will be given.

5 Tests on Simple Functions

Preliminary testing was carried out approximating functions that were themselves linear or piecewise linear. For example, tests were done on the function "2-line", defined as

$$\begin{aligned} y &= 50x + 1000, \quad 0 \leq x < 50 \\ &= 130x - 3000, \quad 50 \leq x < 100. \end{aligned}$$

Parameters for the experiment were the same as pre-

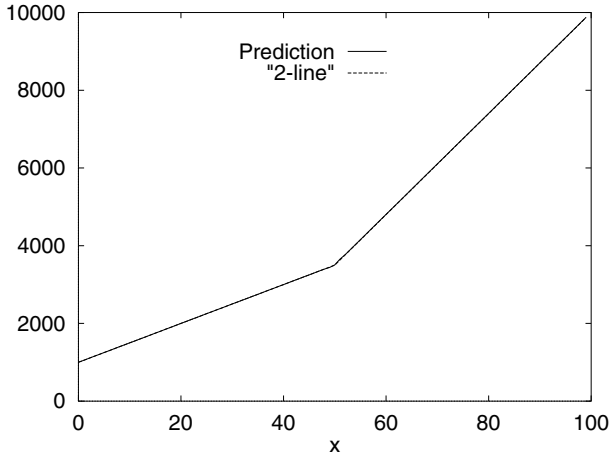

Figure 3: Piecewise-linear approximation to piecewise-linear function "2-line". $\epsilon_0 = 10$.

	CONDITION	PRED	ERR	FITN	NUM
0.	|.....o0000000000|	4705.	19.	.158	62
1.	|.....o0000000000|	4670.	0.	.706	274
2.	|.....o0000000000|	4670.	0.	.044	18
3.	|.....00000000000|	4670.	0.	.090	29
4.	|00000000000o.....|	3057.	16.	.005	6
5.	|00000000000o.....|	3050.	0.	.676	247
6.	|0000000000o.....|	3050.	0.	.377	164

Figure 4: Classifiers from experiment of Figure 3

viously, except for $N = 800$, $\epsilon_0 = 10$, $\theta_{GAsub} = 100$, and $\eta = 0.4$. The approximation obtained (Figure 3) was so close that the plots of the prediction and the function itself are difficult to distinguish visually.

Figure 4 shows the seven classifiers at the end of the run[1]. They are clearly divided into one group for the upper segment of the function and another for the lower segment. The dominant classifier in the upper group, no. 1, has error zero and a predicate covering the interval 52-99 (inclusive). In the lower group, the dominant classifier, no. 5, also has error zero and covers the interval 0-50.

Raising the error threshold caused the approximation to deteriorate. In Figure 5, $\epsilon_0 = 500$. The prediction curve seems to ignore the break at $x = 50$, reflecting the change in slope only gradually. The classifier list showed several that bridged the break point, with predictates from about 30 to 70. Evidently, with a larger error threshold, the system was not forced, as in Figure 3, to evolve classifiers that corresponded closely to the two function segments.

[1]The prediction values represent the most recent weight-vector calculation and are not particularly significant. As in Section 3, this and all following experiments were run for 50,000 input cycles.

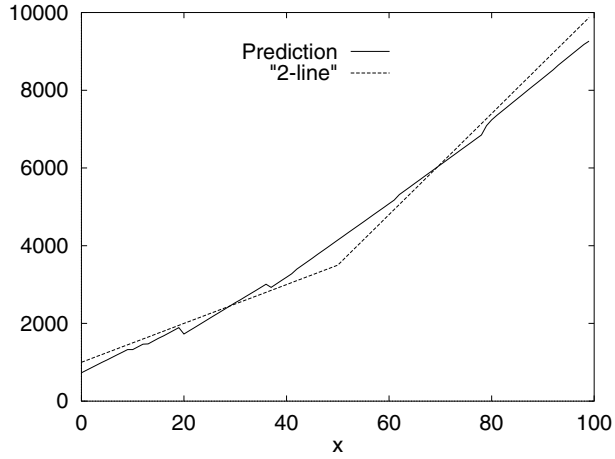

Figure 5: Approximation to "2-line" with $\epsilon_0 = 500$

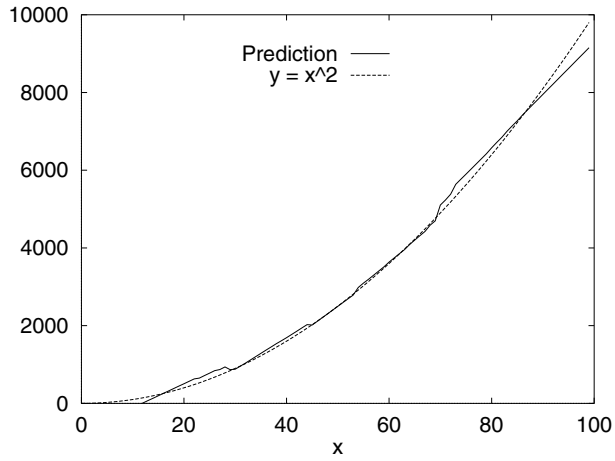

Figure 6: Piecewise-linear approximation to parabola with $\epsilon_0 = 500$.

Figures 6 through 12 show typical results on parabola and sine functions. Parameters were the same as in the "2-line" experiments except $\eta = 0.2$ (an insignificant difference). Values for ϵ_0 are given in the captions. The two values chosen for each function are equivalent to 5% and 1% of the functions' ranges. To highlight the dominant classifiers and save space, the classifier lists include only the highest-fitness classifiers; the full populations are three or four times larger.

While the parabola figures show good linear approximations with a small number of classifiers, it is somewhat surprising that—unlike the piecewise-constant case—the sizes of the interval predicates do not seem to reflect the function's slope. Perhaps for piecewise-linear approximation a different analysis is in order: predicate length may be more related to curve straight-

	CONDITION	PRED	ERR	FITN	NUM
0.	|.........OOOOOOOOOO|	9375.	246.	.286	151
1.	|......OOOOOOOOOOOOO|	9014.	431.	.122	54
2.	|.....oOOOOOOOOOOOOO|	8985.	432.	.176	80
3.	|.....oOOOOOOOOOOOOO|	8906.	456.	.037	37
4.	|....oOOOOOOOOOOOOO|	8744.	456.	.035	33
5.	|OOOOOOOOOOOOOo.....|	4184.	379.	.077	35
6.	|OOOOOOOOOOOOO......|	3756.	247.	.258	122
7.	|OOOOOOOOOOOOo......|	3564.	256.	.066	32
8.	|OOOOOOOOOOOo.......|	3193.	262.	.035	18
9.	|OOOOOOOOOo.........|	2250.	204.	.211	125

Figure 7: High fitness classifiers from experiment of Figure 6.

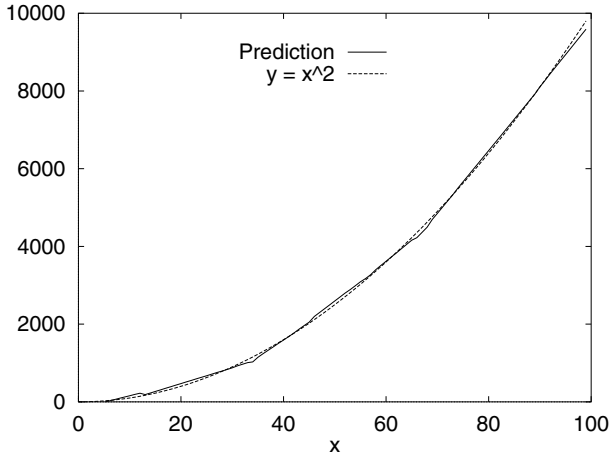

Figure 8: Piecewise-linear approximation to parabola with $\epsilon_0 = 100$.

	CONDITION	PRED	ERR	FITN	NUM
0.	|............oOOOOOO|	9591.	77.	.823	212
1.	|..........oOOOOOO..|	7734.	75.	.081	21
2.	|..........oOOOOOO..|	7687.	89.	.035	9
3.	|......oOOOOOOo......|	4438.	79.	.731	184
4.	|......oOOOOo........|	3149.	26.	.123	41
5.	|..oOOOOOOo..........|	1868.	79.	.228	39
6.	|..oOOOOOo...........|	1621.	61.	.072	13
7.	|OOOOOOO.............|	967.	104.	.607	219
8.	|OOOOOOo.............|	749.	85.	.074	16

Figure 9: High fitness classifiers from experiment of Figure 8.

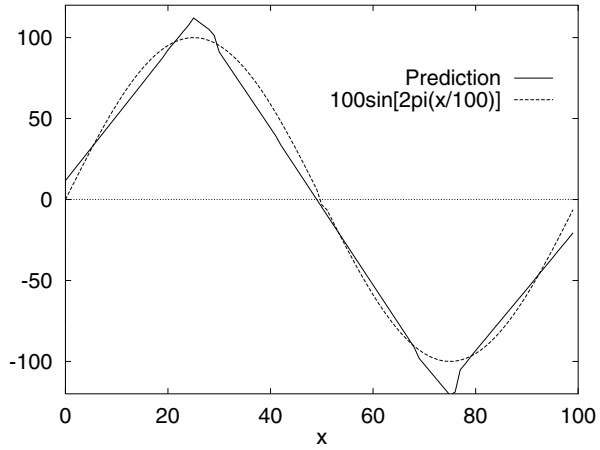

Figure 10: Piecewise-linear approximation to sine function with $\epsilon_0 = 10$.

	CONDITION	PRED	ERR	FITN	NUM
0.	|OOOOOO..............|	120.	10.	.292	76
1.	|OOOOOo..............|	118.	10.	.083	22
2.	|OOOOO...............|	111.	7.	.572	158
3.	|....OOOOo...........|	66.	5.	.032	15
4.	|..............oOOOO|	-20.	4.	.792	173
5.	|.............oOOOOOO|	-23.	12.	.101	78
6.	|.....OOOOOOOOOo....|	-130.	8.	.888	234

Figure 11: High fitness classifiers from experiment of Figure 10.

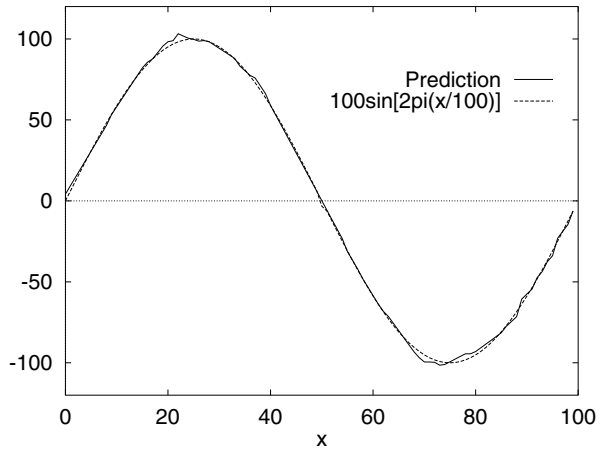

Figure 12: Piecewise-linear approximation to sine function with $\epsilon_0 = 2$.

ness than steepness. For the sinewaves, the approximation overshoots the peaks when ϵ_0 is large (Figure 10), but this effect disappears with smaller values and the curve is quite nicely matched (Figure 12). Figure 11 suggests that the system divides the approximation into classifiers for the beginning, middle, and end of the curve.

6 Multi-dimensional Input

XCSF was initially tested on functions of more than one variable by letting $y = f(x_1, ...x_n)$ be a linear function, i.e., a hyperplane function. The system rapidly evolved solutions with one or a few classifiers and arbitrary accuracy. This was expected, since the classifiers' weight vectors are effectively linear functions. To test XCSF on a multi-dimensional nonlinear function, we chose, somewhat arbitrarily, $y = [(x_1^2 + ... + x_n^2)/n]^{1/2}, 0 \leq x_i < 100$, a sort of "rms" function. Experiments with $n = 3$ went well, so $n = 6$ was tried. Parameters were the same as previously, except $N = 3200$, $\beta = 0.1$, $\theta_{GAsubs} = 200$, and $\eta = 1.0$. The error threshold $\epsilon_0 = 1$ (1% of the range). In contrast to previous experiments in which instances were chosen randomly from a fixed data set, instances were picked randomly from the domain. Figure 13 plots the system error and population size.

Starting initially very high, the system error (a moving average of the absolute difference between XCSF's prediction and the actual function value) fell rapidly to less than 1 (or .01 as plotted on this graph). The population size—in macroclassifiers—rose quickly to about 2400 and stayed there. So the system seemed to have little difficulty approximating the function to within 1%, though quite a few classifiers were required.

7 Classifier Architecture

In XCS, as in other classifier systems, the classifier prediction is a scalar, and the system adapts the classifier conditions and the prediction scalars to find accurate classifiers that are as general as possible. In XCSF, the prediction was replaced by a weight vector computing a linear function, leading to a more powerful and subtle co-adaptation of the condition and the prediction. As an extreme but instructive example, XCSF can approximate a very high-dimensional linear function with $O(1)$ classifiers, far less than required using scalar predictions.

It might be fruitful to extend the concept of classifier structure from the traditional <condition>:<action> ⇒ <prediction> to the more general <condition truth

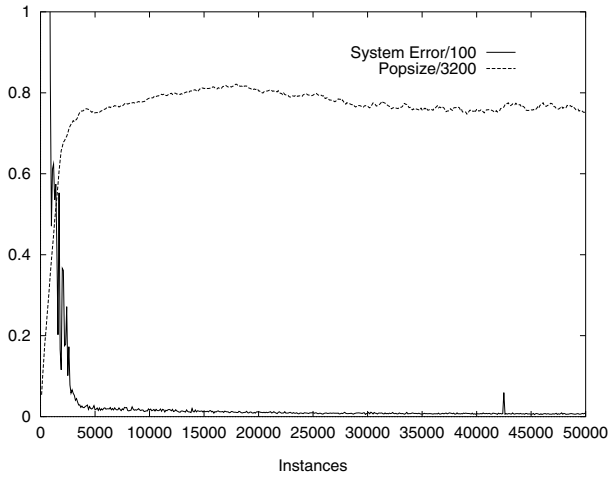

Figure 13: System error/100 and population size/3200 for approximation to six-dimensional "rms" function. $\epsilon_0 = 1$.

function>:<action range> ⇒ <prediction function>. The action range would be a finite interval of effector values, e.g., rudder angle. The condition would be a truth function of x and the prediction a function of x and a, where a is a value in the action range. The two functions and the range would co-adapt. Lanzi (1999) uses condition functions that are Lisp S-expressions. The present paper has investigated linear prediction functions. It is not hard to imagine a classifier that, for a given subspace of x and a finite range of a, predicts the payoff for each x, a combination. Such classifiers could be a step toward classifier systems capable of continuous actions. Even with fixed actions, the possibility of co-adapting condition and prediction functions should lead to more powerful—i.e., accurate and efficient—generalization in classifier systems.

8 Summary and Conclusions

This paper introduced a classifier system, XCSF, designed to learn approximations to functions. The prediction estimation mechanism was used to form the approximations: given an input vector x, the value y of the function to be approximated was treated as a payoff to be learned. In its first incarnation, XCSF produced piecewise-constant approximations. A more advanced version added a weight vector to each classifier, permitting the approximation to be piecewise-linear. Tests on simple one-dimensional functions yielded arbitrarily close approximations, according to the setting of an error parameter. The system tended to evolve classifiers that distributed themselves reasonably efficiently over the function's domain, though some overlap occurred together with the presence of a mod-

erate number of redundant low-fitness classifiers. In limited tests on a six-dimensional nonlinear function, XCSF rapidly formed highly accurate approximations, though the number of classifiers required was much larger than for the one-dimensional functions.

Future work should continue with multi-dimensional functions, to determine the technique's general viability and estimate its complexity in terms of learning time and resources (classifiers) required. Since XCSF approximates linear functions effortlessly, regardless of dimensionality, it is likely that the complexity will relate to the degree of "smoothness" or "flatness" in hyperspace that the function exhibits. Comparisons should be made with fuzzy classifier systems, which appear to be quite different in concept: the output of a fuzzy system is computed jointly by more than one classifier, whereas in XCSF an accurate output can in principal be computed by just one.

Function approximation with XCSF could be useful for on-line learning of any function or mapping from a vector of input values to a function or output value. An example would be financial time-series prediction, where a future price is presumably an approximable function of known prices or other quantities at earlier times in the series.

XCSF's approximation method can perhaps be extended to decision problems in which a decision, 1 or 0, depends on which side of a decision surface an input is on. XCSF would learn an approximation to the decision surface and then a given input's position relative to that surface could be determined.

Piecewise-linear function approximation in XCSF is based on the idea of *calculating* a classifier's prediction, and this leads to the concept of a *generalized classifier* in which the condition is a truth function of the input x and the prediction is a function of x and an action a. Such a classifier would apply in the subspace of the X$\times$ A $\Rightarrow$ P mapping defined by the condition function and an action range specified in the classifier. The action range could be continuous, permitting selection of the best action value by maximizing the payoff over the range, a possible step toward classifier systems with continuous actions.

Acknowledgements

The author appreciates the comments of four anonymous reviewers, and has responded as space allowed. This work was partially supported by NuTech Solutions Inc.

References

Bonarini, A. (2000). An Introduction to Learning Fuzzy Classifier Systems. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson (Eds.), *Learning Classifier Systems. From Foundations to Applications*, Volume 1813 of *LNAI*, Berlin, pp. 83–104. Springer-Verlag.

Butz, M. V. and S. W. Wilson (2001). An Algorithmic Description of XCS. See Lanzi, Stolzmann, and Wilson (2001).

Lanzi, P. L. (1999). Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pp. 345–352. Morgan Kaufmann: San Francisco, CA.

Lanzi, P. L., W. Stolzmann, and S. W. Wilson (Eds.) (2001). *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS-2000)*. Springer-Verlag.

Mitchell, T. M. (1997). *Machine Learning*. Boston, MA: WCB/McGraw Hill.

Valenzuela-Rendón, M. (1991). The Fuzzy Classifier System: a Classifier System for Continuously Varying Variables. In *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA91)*, pp. 346–353.

Widrow, B. and M. E. Hoff (1988). Adaptive switching circuits. In J. A. Anderson and E. Rosenfeld (Eds.), *Neurocomputing: Foundations of Research*, pp. 126–134. Cambridge, MA: The MIT Press.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation 3*(2), 149–175.

Wilson, S. W. (1998). Generalization in the XCS classifier system. 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*, pp. 665–674. Morgan Kaufmann: San Francisco, CA.

Wilson, S. W. (2001). Mining Oblique Data with XCS. See Lanzi, Stolzmann, and Wilson (2001).