X-TCS: Accuracy-based Learning Classifier System Robotics

Matthew Studley & Larry Bull
School of Computer Science
University of the West of England
Bristol BS16 1QY, U.K.
matthew2.studley@uwe.ac.uk

Abstract- Most research in the field of Learning Classifier Systems today concentrates on the accuracy-based XCS. This paper presents initial results from an extension of XCS that operates in continuous environments on a physical robot. This is compared with a similar extension based upon the simpler ZCS. The new system is shown to be capable of near optimal performance in a simple robotic task. To the best of our knowledge, this is the first application of an accuracy-based LCS to controlling a physical agent in the real world without *a priori* discretization.

1 Introduction

Learning Classifier Systems (LCS) (Holland and Reitman, 1978) offer many advantages for use in robotic learning. They have an inbuilt ability to generalize, the rule-bases evolved can be readily interpreted, and they may converge on optimal behaviour faster than purely evolutionary approaches. The inbuilt generalization can be used to avoid the need to decide *a priori* an appropriate level of granularity of actions or events.

This paper describes an extension of the popular accuracy-based LCS, XCS. The new system, X-TCS, is capable of learning to perform a simple robotic task in a near optimal fashion. Learning is all online, with no simulation. Little parameter adjustment is necessary to achieve these results, suggesting that the system may easily be more generally applied.

2 Learning Classifier Systems

An LCS is a parallel production rule system, which uses reinforcement learning technique to change the fitness of the rules contained in the system and a genetic algorithm to search the rule base for new useful and *general* rules through a combination of genetic search and reinforcement learning.

A number of different LCS models have been described, stemming from Holland's original work. Wilson (Wilson, 1994) introduced the 'Zeroth Level Classifier System', ZCS. In this he made a number of simplifications to make the system easier to understand and to implement.

ZCS contains a population [P] of N classifiers. In Wilson's original formulation, classifiers have a condition and action part encoded using the ternary and binary alphabets with the wildcard '#' allowing generalization. Upon presentation of the binary encoding of the environment, the subset of [P] which matches is termed the 'match set' [M].

It may happen that no classifier in [P] matches the environmental input, and [M] is empty. A classifier is created *de novo* by a process called *cover*. Its condition matches the current environmental input, each bit having a probability $P_{\#}$ of being replaced by a wildcard character. An action is chosen at random from the set of possible actions. The new classifier is assigned the average fitness (or 'strength) of the population, and replaces a classifier chosen from the population with a probability inversely proportionate to fitness. Cover may also take place if the total fitness of [M] is less than a certain pre-specified fraction ϕ of the mean fitness of [P].

One action must be chosen from the set of actions advocated by the classifiers in [M]. Roulette selection on the fitnesses of the classifiers in [M] is commonly used, such that classifiers of higher fitness are more likely to be selected. All the classifiers in [M] which advocate the same action form the *action set* [A]. The chosen action is then taken.

Reinforcement in ZCS occurs through redistributing reward through the action sets that lead to this reward. A percentage β of the fitness of the classifiers in [A] is deducted and placed in an initially empty 'bucket'. After the action has been taken, the fraction β of any external reward received is divided amongst the classifiers in [A]. A record is kept of the previous action set $[A]_{-1}$, and if this is not empty, the 'bucket' is discounted by λ and the then divided equally between its members. In this way a discounted reward flows back through the 'implicit bucket brigade'. Additionally, the classifiers which were in [M] but not in [A] have their fitnesses reduced by a fixed fraction τ the tax rate. This causes classifiers which advocated a different action from the one taken to be penalized. The interaction between the flow of reinforcement signals from the environment and tax rate will effect the balance of exploitation and exploration in the learning system, though it should be noted that the likelihood of generation of new classifiers by cover and through genetic churn will also interact with this balance.

The 'implicit bucket brigade' algorithm outlined above was noted by Wilson to be similar in some respects to Q-learning (Watkins, 1989). In contrast to Tabular Q-learning, where a Q value is derived for every state-action pair in the environment, an LCS allows the generation of rules that generalize over many states. In Reinforcement Learning, function approximation techniques such as artificial neural networks are used for similar reasons; see Sutton and Barto (1998) for an overview.

0-7803-9363-5/05/$20.00 ©2005 IEEE.

In addition to the adjustment of fitness through the process outlined above, there is a panmictic genetic algorithm (GA) that replaces weak classifiers with copies of stronger classifiers. Parents and replaceable classifiers are once again chosen using fitness proportionate selection. The new classifiers generated to replace the weak may undergo single point crossover with the probability χ and may mutate at any position in the condition or action part with the probability of μ per allele. The GA is invoked after all reinforcement has taken place with the fixed probability ρ. New classifiers generated by the GA are initialised at half their parent's fitness, and the parent has its fitness reduced by a half. Therefore reproduction (in the absence of the genetic operators) does not in itself change the likelihood that an action will be chosen.

Wilson suggested that ZCS was incapable of solving simple maze tasks due to failure in balancing exploration and exploitation; in short, the algorithm has found a sub-optimal solution due to an incomplete exploration of the problem space, and fixated prematurely upon a solution. Subsequently, it was shown by Bull and Hurst (2002) that ZCS was capable of solving these problems given suitable parameter values. However, tuning the parameters for ZCS is a difficult task. Some cannot be adjusted in isolation, and, although self-adaptation (Bull et al, 2000) of some parameters has been shown to be effective, this is not the case of the reinforcement learning parameters which are crucial to the ability of the learner to achieve optimal behaviour in multi-step problems such as those of robot control.

Believing ZCS incapable of optimality, Wilson introduced XCS, which differs from ZCS in many ways, most importantly, that fitness is proportionate to the *accuracy* of the classifier's prediction, and that the GA does not choose parents from the population as a whole, but instead from members of action sets. The intention behind XCS is that the system should form a set of rules that provide a complete and accurate maximally general map of the space of state-action pairs.

In addition to its condition and an associated action, each XCS classifier also records: p, the predicted value of taking this action, ε, the prediction error, *exp*, which records the number of times the classifier has been in the 'action set', [A], *ts*, a time-stamp that records when the classifier was last in [A] when the GA operated, *as*, which keeps an estimate of the average size of [A] containing this classifier, n, the 'numerosity', records the number of individuals in the population represented by this 'macroclassifier' (Macroclassifiers are treated as if they are n copies of identical classifiers, providing a computational advantage.), and F, the classifier's fitness.

Once [M] has been formed, XCS then forms a prediction $P(a_i)$ of the payoff expected from each possible action. These values are stored in the Prediction Array *PA*. Some of these values will be null if no member of [M] advocates this action. The system prediction for an action is a fitness-weighted average of the predictions of all classifiers in [M] which advocate that action.

An action is chosen from the set of actions suggested by the classifiers in [M]. In typical implementations, a balance between 'exploration' and 'exploitation' is represented by performing half the trials with a random action selection policy, and the other half with a deterministic policy. One action is selected and an action set [A] is formed. The chosen action is performed, and any external reward is returned by the environment.

The reinforcement component in XCS consists of modifying the following parameters associated with each classifier, in the following order: *exp, p,* ε and F. *exp* is incremented to record the number of times this classifier has appeared in an [A]. p and ε are adjusted using the Widrow-Hoff procedure with the learning rate parameter β ($0 < \beta \leq 1$) only after a classifier has been adjusted at least $1/\beta$ times, otherwise the new value is simply the average of the previous value and the current one. This closely resembles the Q-learning technique.

In the final step of the reinforcement process, each classifier in the set has its fitness F adjusted, dependent on the relative accuracy of the classifier within [A].

As noted above, the second main area in which XCS differs from ZCS is in the application of the GA. Wilson was motivated to develop XCS by the belief that ZCS failed to achieve optimality due to the emergence of over-general rules; rules which are good in one niche and achieve a high fitness value are chosen over more optimal rules in a different niche where they also match, but in which they are of lower utility. In XCS the GA is restricted in its choice of parents to [A], and thus there is exploration of the space of possible generalizations containing a specific state-action pair. Since classifiers that participate in more niches (and are accurate) are rewarded more often and have more opportunities to replicate and search the space of generalizations through the genetic operators, this introduces a pressure towards the generation of **general** classifiers *for each state-action pair explored by the system.*

The 'triggered-niche' GA used by XCS in this implementation works in the following way. Each classifier has an associated measure of the number of time steps since it was last involved in an application of the GA. The GA is invoked if the average time period since the classifiers currently in [A] were last subject to the GA is greater than a threshold value θ_{GA}. Two parents are selected by a fitness-proportionate method (roulette selection) and after possible two-point crossover and mutation, are inserted into the population, replacing other classifiers.

There are two forms of 'subsumption' in XCS that contribute to computational efficiency at the expense of a decrease in diversity in the population. In 'GA subsumption' an offspring that has a condition already represented by a more general and experienced parent is not added to the system, but the parent's 'numerosity' is incremented. In 'Action Set subsumption', the most general classifier in [A] is found, and all other classifiers in [A] are tested against it to see whether it subsumes their condition. If it does, they are discarded and the most general classifier has its numerosity increased by one as before.

3 Evolutionary Computing, Reinforcement Learning and Classifier Systems in Robotics.

Evolutionary techniques have been shown to be effective for developing robotic controllers. Cliff et al. (1993) reported the evolution of vision systems. Floreano and Mondada (1996) used a GA to evolve a population of strings of floating-point numbers representing weights and threshold values of a discrete time recurrent neural network topology. Matellan et al. (1998) evolved fuzzy controllers for a Khepera to perform navigation and obstacle avoidance tasks. Marocco and Floreano (2002)used an evolutionary approach to evolve an active vision system for a Koala robot. However, evolution requires the assessment of the fitness of many phenotypes. If this requires the phenotypes to be embodied, the process is slow. Speed-up can be achieved by the use of simulation, e.g., (Jakobi et al., 1995), (Nolfi et al., 1994), but this runs counter to the philosophical and pragmatic reasons for evolving the controllers.

A number of researchers have used reinforcement learning on robotic platforms. Mahadevan and Connell (1991) used Q-learning to control a robot in a box pushing task, applying an *a priori* discretisation to render the environment into 18 state bits. Since the authors report some perceptual aliasing, this imposed discretisation clearly hides potentially important environmental information from the learner. A similar approach of generalising through the *a priori* discretisation of continuous data is used by Asada et al. (1996). The environmental input from a video camera is simplified into discrete categories before being presented to the learner. Once an action has been chosen, the robot continues to take that action until the environmental state changes, at which point another action is chosen and the action value function is updated. In this way an action always causes the transition from one state to another.

In both the work reported by Mahadevan and Connell, and Asada et al, the world has been discretised according to an arbitrary scheme invented by the experimenters. This approach, while effective in allowing the learner to build a table of values mapping discrete states to actions, limits the learner's ability to deal with different environments. The discretisation may need to be changed (by the experimenter according to his specialist understanding of the new environment and learner) to learn the same task in a bigger room. If the discretisation is too course-grained then optimality cannot be achieved, but since Q-values must be established for every combination of state and action a fine-grained discretisation will result in having to build a huge map of state-action values, thus impeding learning. The memory requirements grow, and inefficient use is made of experience – since each state-action value must be estimated in isolation, no use is made of the fact that states near to each other are likely to have similar values and similarly optimal actions (Kaelbling, 1996).

In order to address the problem of the combinatorial explosion of state-action values that must be learned, *function approximators* are widely used to generalize over the state-action space. Many techniques have been applied, including neural networks, fuzzy logic, and CMAC

Santamaria et al. (1998) demonstrate the use of CMAC, or Cerebellar Model Articulation Controller, in which each input activates some subset of overlapping tiles of the state-action space. The predicted Q-value is the sum of the values represented by these 'features'. Clearly, the granularity of the tiling controls the generalising abilities, and resource consumption, of the function approximator.

Thrun and Schwartz (1993) make the point that since a function approximator introduces some noise due to over-generalization, when combined with a recursive value estimation scheme this may preclude the learner from achieving optimality for certain values of λ. Smart and Kaelbling (2000) show one solution to this problem in their HEDGER learning algorithm that generates a prediction of Q-values based upon similar past experience. They report rapid learning which approaches the best results achieved under human control.

In summary, evolutionary methods generally take many hours to produce robotic controllers, unless partially or totally trained in simulation (or trained in real-time by a human operator). Reinforcement learning techniques commonly require the discretisation of the input space, which may compound the problem of perceptual aliasing.

Some researchers have presented results using LCS, which combine both evolutionary and reinforcement learning. Cliff and Ross (1994) discuss results in simulation using ZCS in a simulated environment with continuous space and discrete actions of fixed size and Stolzmann (1999) presented results using a simulated Khepera to explore latent learning with the Anticipatory Classifier System (ACS) in which the environment was rendered into discrete states before presentation to the classifier system, but comparatively few accounts have been published on the application of LCS to the control of physical robots in the real world.

The most widely cited example of robot-based LCS learning is that of Dorigo and Colombetti (1998). In this work they present a novel LCS, based upon Holland's original formulation. The architecture produced by Dorigo and Colombetti closely resembles that of Connell's colony-style subsumption architecture as used in Mahadevan and Connell's paper above.

Dorigo and Colombetti apply this parallelised, hierarchical system to a number of real robot control problems of varying complexity. They use two training policies, either to reward the result (i.e. give reward when some goal is achieved), or to reward "the intention". In the latter, small rewards are given when an action is taken that is consistent with a predefined model of the 'correct' state-action map. While this work is impressive, a number of points should be made:

Firstly, there is considerable reliance on *a priori* knowledge of the problem. This informs the hierarchical decomposition into behavioural modules, and the training policy used to reward the intention. Secondly, discrete inputs and actions are used. In effect, the robot is operating in a noisy grid-world. Once again, *a priori*

knowledge must be used to establish the granularity of the grid in which the robot is operating – how big is a step forward?

Additionally, the use of a reward the intention policy which aids learning by providing continual reinforcement requires the experimenter to provide a training program that can assess the quality of each action taken by the learner. For some problems this could be easy to specify, however, this relies on much more knowledge than a scheme based upon delayed reinforcements. As the task becomes more complex, the reinforcement program will become more difficult to write, prone to error and therefore could result in the learning of behaviours which, rather than solving the overall task, profit from exploiting unforeseen local minima.

Bonarini et al. (1997) detail a novel system called ELF, a controller based upon the evolution of fuzzy rules. Since rules that are triggered at the same time have the same antecedent, they compete with each other. Rules with different antecedents are triggered at different times, and therefore may co-operate together. Rather than evolving many complete FLCs – an approach they liken to Pittsburgh classifier systems, and which they ignore as unfeasible for evolving real robots – or evolving individual rules in a panmictic GA as used in Michigan classifier systems, they instead partition rules with the same antecedents together considering them to belong to the same sub-population, and run their evolutionary algorithm on each of these subsets in isolation. This is similar to the niche GA as used in XCS. If there exists too many rules in an ELF sub-population, the worst are deleted, providing a fitness-proportionate selective pressure.

ELF uses a cover operator, and reinforcement with discounting in a similar fashion to Q-learning. Reinforcement is carried out at the end of an 'episode' – a number of control cycles. The number of control cycles in an episode can be predefined, or can be triggered upon reaching a certain state. During an episode, only one rule in each sub-population can fire, and this rule is chosen randomly. Reinforcement is proportional "*...to a rule's contribution to the obtained result*". It also reinforces with discounting the rules that triggered in past episodes.

The ELF algorithm is an interesting approach that attempts to solve a number of specific challenges. It uses real-valued input, and in the animat experiments using the 'FAMOUSE' agent, discrete actions (the consequents are limited to seven angles evenly distributed in the range – 180° to +180° for steering). The authors describe good results in a task to follow a light. Later extensions to the algorithm were tested on a physical robot called 'CAT'.

Katagami and Yamada report work (2000, 2001) in which they attempt to speed learning by bootstrapping an LCS through interactive human training. The environment, as perceived by the robot, is presented to the operator who can guide the actions of the robot using a joystick. Their LCS is based upon XCS. The learner performs alternating taught and autonomous trials. They present results in simulation which show an improvement in the speed of learning with teaching, compared with the LCS alone, and experiments in wall-following and multi-agent robot soccer using a real Khepera robot.

Ideally, a mechanism for producing robot controllers should be capable of the following:

- Life-time learning to enable the controller to adapt to changes in its environment.
- The world is its own best model. Simulation should not be necessary.
- The system should be able to deal with delayed reinforcement. Complex problems that must be solved by the composition of behaviours may not be easily represented by a reward the intention system of continual assessment.
- It should not take excessive time to produce optimal, or near optimal, behaviour.
- It should not be necessary for the experimenter to decide *a priori* schemes for reducing the environment's complexity, since such schemes may be inaccurate, hide vital environmental cues, and may stop the controller from adapting to changes in its environment. Therefore neither the input or output space should be manually discretised.

4 The Temporal Classifier System (TCS)

Hurst et al. presented work in which they applied two classifier systems to problems of robotic control (Hurst et al, 2002, Hurst 2003). They first demonstrate that ZCS can be used successfully to learn an obstacle avoidance behaviour on a LinuxBot (Winfield, 2003). In order to define automatically what constitutes an 'event', the action-selection sub-system of ZCS was extended. Initially, a match set is formed in the same way as in ZCS, and thence an action set. The action is taken. In a new 'drop decision' cycle, the environmental representation is repeatedly presented to the members of [A]. If a time-out value has been reached, or an external reward has been received, the drop decision cycle terminates. In the case that the timeout value has been reached, the members of [A] are not assigned to the previous action set $[A_{-1}]$ so they receive no update in the next cycle. If an external reward is achieved, the members of [A] are rewarded, and the next cycle of sense, [M] formation, etc. starts.

If neither of these conditions is true then a decision is made on whether the classifiers in [A] continue to match the current environment. Two sets are formed, the 'drop set' of classifiers in [A] which no longer match, and the 'continue set' of classifiers which do.

If no classifier matches the current input, i.e. [continue] is empty, then the drop decision cycle ends, and the familiar stages of reinforcement (and optionally the GA) are performed, $[A_{-1}]$ being set to [A].

If all classifiers match the current input, i.e. [drop] is the null set, the cycle continues, and thus the original action carries on.

If some classifiers are in [drop], and some are in [continue], a decision must be made on whether to continue or terminate the current action. This decision is

made on the fitnesses of the classifiers, and may be done stochastically using e.g. roulette-wheel selection on [A] to pick a classifier, or deterministically by picking the strongest. If the classifier thus chosen is in [continue], [A] is set equal to [continue]. The classifiers in [drop] will not receive an update via the bucket brigade. If the picked classifier is in [drop], [A] is set equal to [drop], the classifiers in [continue] will be excluded from the bucket brigade updates, the current action is terminated and the processes of reinforcement etc. take place.

Hurst et al.'s TCS – a Temporal Classifier System – also incorporates an approach that tackles Semi-Markov Decision Problems within the Reinforcement Learning framework. This addresses a second problem; how to choose between actions that move the learner from one state to the next, but take different amounts of time to complete.

Hurst et al. incorporated these changes to the bucket-brigade into ZCS in the following ways. Firstly, the external reward is discounted according to the amount of time taken to reach the goal, t^i. This will favour efficient over-all solutions.

$$r = e^{-\sigma t^i} r \tag{3}$$

Secondly, the discount factor γ also factors in time, but in this case the time taken to make the transition *between* events, t^i. This will favour actions that transition between states efficiently.

$$\gamma = e^{-\eta t^i} \tag{4}$$

In effect, σ and η are different learning rates that change the emphasis placed upon the overall time to achieve external reinforcement, and the time taken performing individual actions.

The original ZCS update algorithm was;

$$S_{[A]} \xleftarrow{\beta} r_{imm} + \gamma S_{[A]'} \tag{5}$$

in which $S_{[A]}$ is the current action set, r_{imm} is the immediate reward, $S_{[A]'}$ is the previous action set, and $\xleftarrow{\beta}$ is the Widrow Hoff gradient descent procedure with the learning rate β. Incorporating (3) and (4) into (5) gives the update procedure that Hurst et al used in TCS;

$$S_{[A]} \xleftarrow{\beta} e^{-\sigma t^i} r_{imm} + e^{-\eta t^i} S_{[A]'} \tag{6}$$

In summary then, there are now two forms of discounting; external reward is dependent on the total time taken to achieve the goal, and the amount of reinforcement flowing to previous action sets is proportional to time, favoring longer actions over short ones.

4.1 Experimental Details

The hardware platform used for the experiments reported here is the same type as that used by Hurst et al., that is, a 'LinuxBot'. In the experiments presented here, the robot was equipped with three IR proximity detectors, or 'IR bumpers', one facing forward, one forward and slightly to the left, and the third slightly to the right. Two physical bumpers are mounted at the front of the robot, one wrapping around to the left side, and one to the right. The robot is equipped with an elevated mast on which three light-dependent resistors (LDRs) are mounted. Again, one faces forward, one is angled 45° to the left, and one 45° to the right.

The environment consists of a square enclosure, measuring approximately 300 cm on each side. The walls of the enclosure are painted matte black. There is a light source (halogen lamps) at one end of the enclosure, approximately half way between the two sides. All tiles of the floor supply power to the robot via pickups, but two of the tiles are powered by their own power supply, which supplies no other tile. By monitoring the current drain, a PC can signal to the LinuxBot when it is on one of these tiles via the wireless LAN. One tile to the side of the lights is considered the goal at which external reinforcement of 1000 is applied, another tile at the opposite side the home tile from which trials start.

No special measures were taken to insulate the environment. The arena is open to natural diffuse light, and is situated next to a busy corridor. Any learning must therefore cope with quite a high potential of noise, in addition to the noise introduced by the LDR sensors themselves.

TCS is presented with an environmental input consisting of three real numbers representing the resistance of the LDR sensors, scaled between 0.2 and 0.8. The condition part of the classifiers is encoded as un-ordered pairs of real numbers in the range [0, 1], one pair for each environmental input. A pair is considered to match the corresponding input value if one of the pair is smaller or equal to the target, and the other is larger or equal. The action of the classifier is an integer, representing the actions move forward, turn continuously to the left, and to the right. Note that these actions are continuous; there is no artificially imposed granularity in the action space.

An unordered representation was chosen for the condition pairs based upon the work reported by Stone and Bull (2003). In extending XCS for use in real valued environments, Wilson has suggested two representations; 'Centre-spread' (Wilson, 2000) and 'Lower-Upper Bound' (Wilson, 2001). Stone and Bull show that the 'Centre-spread' representation introduces significant bias, and point out that the 'Lower-Upper Bound' representation demands that any operation that could result in $c_1 > c_2$ must result in a reordering of the alleles. To simplify this, they introduced the 'Unordered Bound Representation' as used here, which they show has the same desirable properties as the 'Lower-Upper Bound' scheme.

The crossover and mutation operators are altered from those in ZCS to deal with the new representation. In crossover, there is an equal chance of crossing over at any point in the condition. Mutation can occur at any point in the classifier with probability μ. Mutation in the case of the real numbers of the condition is effected either the addition or subtraction of either a small number drawn from a Gaussian distribution centred on the current value, or a fixed small change (here 0.005). Action mutation is by picking an integer from the set {0,1,2} at random, such that the chosen action is different from the current one.

In the initial population, classifier conditions are created randomly in the range [0,1]. During cover, the current environmental input e is used as a centre and two values are created in the range $[e\text{-}C_{max}, e\text{+}C_{max}]$, where C_{max} is 0.2. To speed convergence, we select deterministically the classifier from [A] which determines whether the drop decision terminates, rather than using roulette-wheel selection. In addition to the conditions of the drop decision described by Hurst et al, we also terminate the drop-decision cycle if the bumpers are on. In this case the *last* drop-set is returned as the set which will receive reinforcement to discourage the robot from walking into obstacles. If there is no 'last drop-set', the null set is returned.

4.2 A 'Trial'

In the experiments described here, the system is first calibrated on the light readings immediately next to the light source, and pointing in the opposite direction. To reduce the impact of noise from the LDR sensors, these readings are the average of 100. The robot is then placed on the 'home' tile, and the first learning trial begins. TCS is stopped when the robot reaches an appropriate 'goal' tile. The learning algorithm is then stopped, and the robot is placed under the control of an obstacle avoidance algorithm, 'bouncing' around the experimental pen until the robot is on the 'home' tile. The robot orientates itself approximately towards the light, and the next trial begins. In order to save time, any trial that lasts for longer than 30 seconds is considered to have 'timed out' – there is no reinforcement, and the robot returns to start a new trial as above.

The robot may start at any point on the 'home tile', and so the distance to the goal may vary between trials by approximately 80 cm. Also, delays in response from the power supplies may mean that the robot is on the goal tile for up to a second before this fact is signalled to TCS. These two factors mean that the 'optimal' time to complete a trial should be considered as a range of values; between six seconds and nine seconds. The following parameter settings were used on the single objective task. N 600, S_0 10, β 0.2, σ 0.1, η 0.7, τ 0.1, χ 0.5, μ 0.05.

TCS was found to easily achieve good results on the single objective task, comparable to those reported by Hurst (2003). In Figs. 1 and 2 we show the average results of five runs, each run lasting for 200 trials. Fig. 1 shows the ratio of failures (a trial timed out) against success. TCS is soon capable of performing nearly optimally, finding the goal each time with only an occasional failure. In Fig. 2 the upper trace shows the windowed average of the percentage success, windowed over 50 trials, and the lower trace shows the average time taken to reach the goal. After a short initial period in which performance fluctuates, the algorithm quickly shows that it is capable of nearly optimal performance. There is a strong trend for the time taken in each trial to reach the goal to be within the range we consider optimal.

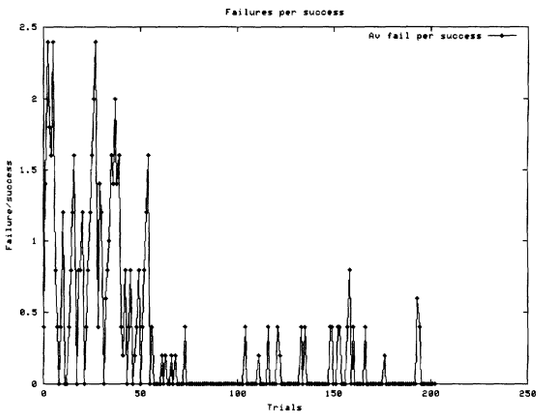

Figure 1: TCS Failure to Success ratio

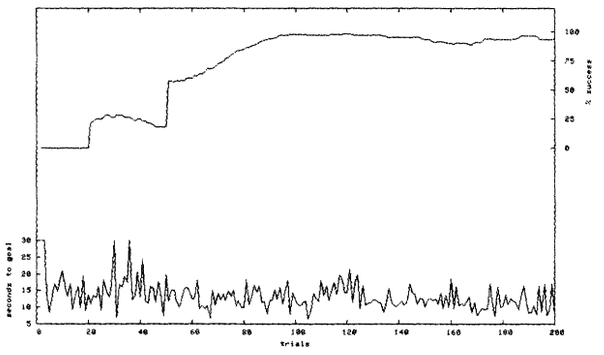

Figure 2: TCS Percentage Success and time to goal

It should be noted that the settings of the reinforcement parameters were based upon those reported by Hurst et al in their 'towards the light' experiment. The system may have achieved better or worse results with different settings.

5 An Accuracy-based Temporal Classifier System (X-TCS)

In contrast to the parameter sensitivity of ZCS, the accuracy-based XCS requires little adjustment of its reinforcement learning parameters in order to achieve optimality in the same simulated tasks. We have suggested (Studley & Bull, 2005) that XCS could be made suitable for online control of a robot by substituting a probabilistic action selection policy – in this case, a roulette-wheel mechanism - for the random action selection policy typically used in 'explore' trials, and noted that there is some evidence that using such a policy may enable better performance with smaller population sizes as might be necessary in a resource-limited environment such as a physical robot.

The changes made to XCS to produce X-TCS are a parallel of those described for TCS. XCS forms a match set, and thence an action set using either a deterministic exploit policy or a roulette-wheel explore policy. The action is then initiated and the algorithm enters the drop

decision cycle, in which it is determined whether to carry on with the current action or stop and create a new match set. The process continues until external reinforcement is attained or some time-out value for the trial is reached. Rather than basing the drop decision on the fitness of the classifiers in drop and continue sets, X-TCS uses the values in the prediction array. In this way the drop decision is reward-based as is the fitness-based decision in TCS.

In addition to the use of the drop decision cycle to determine automatically what granularity of environmental change may be appropriately discovered, the update procedures of reinforcement learning are also modified. As in TCS, the external reward is discounted according to the amount of time taken to reach the goal, and the reinforcement that flows from the classifiers that advocate an action to their predecessors is similarly discounted. As before, these alterations should favour efficiency of the solution as a whole, and the use of a smaller number of long actions rather than a large number of small actions.

The same changes are made to the representation of the classifiers' condition and hence to the genetic operators as are described for TCS. Although GA subsumption is used to compact the rule base and promote generalization, action set subsumption is not used since this would remove the variability in the action set upon which the drop decision cycle relies. The XCS policy of alternating explore and exploit trials was used, but to show the online performance of X-TCS we show the average of both explore and exploit trials, rather than only the deterministic exploit trials performance as is commonplace. All results are the average of five experiments. The total number of trials performed is comparable to that presented for TCS, i.e. where 200 TCS trials were performed, 100 explore and 100 exploit trials were performed with X-TCS. Where a windowed average is shown, the window size is therefore reduced to 20 trials rather than 50 as used for the TCS results.

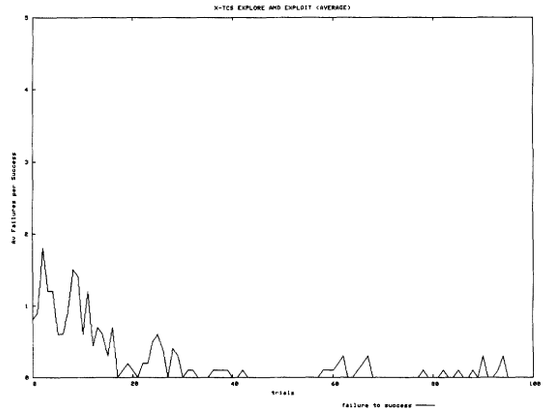

Figure 3: X-TCS Failure to Success ratio

Fig. 3 shows the failure to success ratio of X-TCS. Fig. 4 shows the time taken to reach the goal decreasing towards the optimum band, and Fig. 5 shows percentage success as a windowed average increasing to nearly 100%. The graphs are broadly comparable to those of TCS. There was a small but significant difference in the number of actions taken, with X-TCS tending to take slightly more actions to reach the goal (not shown).

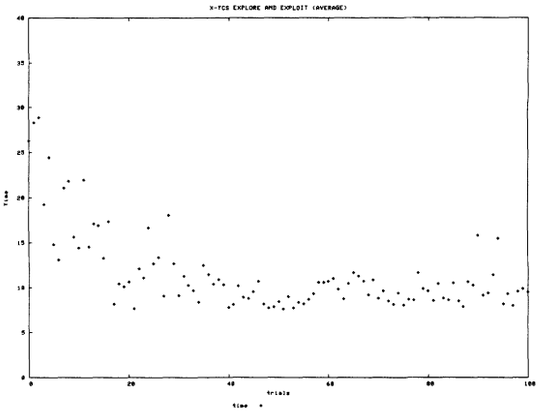

Figure 4: X-TCS Time to goal

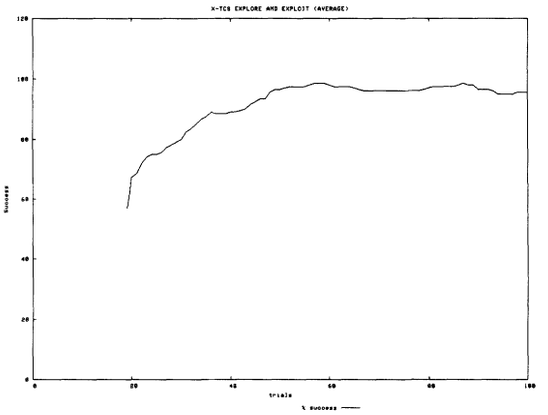

Figure 5: X-TCS Percentage Success

All parameter settings are as used for TCS. Where XCS-specific parameters were used these are as reported in Wilson's original paper (Wilson, 1995).

6 Conclusions and Further Work

TCS incorporated two major changes to the fitness-based classifier ZCS; the use of the drop-decision cycle to automatically determine the granularity of events appropriate to the problem, and the use of temporal discounting to favour few long actions and efficient timely solutions. We have made the same changes to the accuracy-based XCS, basing the drop-decision cycle on the prediction array in keeping with XCS' philosophy. The new system, X-TCS has shown itself capable of near optimal online performance that equals that of TCS.

In simulation, it has been seen that ZCS requires careful tuning of the reinforcement learning parameters to achieve optimal performance, which does not appear to be true of XCS. We will investigate more complex problems with multiple goal states using both systems. We expect that where we are unable to demonstrate optimal performance with TCS, X-TCS will still perform optimally. X-TCS requires little parameter adjustment, learns relatively rapidly, and appears well suited to the online control of a real robot for these simple tasks.

7 References

Asada, M., Noda, S., Tawaratumida, S., & Hosoda, K. (1996) "Purposive behavior acquisition for a real robot by vision-based reinforcement learning." *Machine Learning*, 23:279—303.

Bonarini, A., & Basso F. (1997) "Learning to compose fuzzy behaviors for autonomous agents". *International Journal on Approximate Reasoning*, vol. 17, no. 4.

Bull, L., Hurst, J. & Tomlinson, A. (2000) "Self-Adaptive Mutation in Classifier System Controllers." In J-A. Meyer, A. Berthoz, D.Floreano, H. Roitblatt & S.W. Wilson (eds) *From Animals to Animats 6 - The Sixth International Conf. on the Simulation of Adaptive Behaviour,* MIT Press.

Bull, L. & Hurst, J. (2002) "ZCS Redux." *Evolutionary Computation* 10(2): 185-205.

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

Cliff, D. T., Harvey, I., & Husbands, P. (1993). "Explorations in Evolutionary Robotics." *Adaptive Behaviour*, 2:73--110.

Dorigo, M. & Colombetti, M. (1998) *Robot Shaping An Experiment in Behavior Engineering,* Intelligent Robotics and Autonomous Agents series, vol. 2. MIT Press.

Floreano, D. & Mondada, F. (1996). "Evolution of homing navigation in a real mobile robot." *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:396--407.

Holland, J.H. & Reitman, J. (1978). "Classifier Systems based on Adaptive Algorithms", *Pattern Directed Inference Systems*, Waterman, D.A. & Hayes-Roth, F. (eds), Academic Press, NY, pp 313-329.

Hurst, J., Bull, L. & Melhuish, C. (2002) "TCS Learning Classifier System Controller on a Real Robot." In J.J. Merelo, P. Adamidis, H-G. Beyer, J-L. Fernandez-Villacanas & H-P. Schwefel (eds) *Parallel Problem Solving from Nature - PPSN VII.* pp588-600.

Hurst, J. (2003) *Learning Classifier Systems in Robotic Environments* PhD Thesis, Faculty of Computing, Engineering, and the Mathematical Sciences, University of the West of England, January 2003.

Jakobi, N, Husbands, P. & Harvey, I. (1995) Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Moran, A. Moreno, J. Merelo, & P. Chacon, editors, *Advances in Artificial Life: Proc. 3rd European Conf. on Artificial Life*, 704--720. Lecture Notes in Artificial Intelligence 929.

Kaelbling, L.P., Littman, L.M. & Moore, A.W. (1996) "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237—285.

Katagami, D. & Yamada, S. (2000) "Interactive Classifier System for Real Robot Learning", *IEEE International Workshop on Robot-Human Interaction (ROMAN-2000)*, pp.258-263

Katagami, D. & Yamada, S. (2001) "Real Robot Learning with Human Teaching", *The 4th Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems*, pp.263-270.

Mahadevan, S. & Connell, J. H. (1991) "Automatic Programming of Behaviour-based Robots Using Reinforcement Learning", *Artificial Intelligence*, Vol. 55, 311-365.

Marocco, D., & Floreano, D. (2002), "Active Vision and Feature Selection in Evolutionary Behavioral Systems", In B. Hallam, D. Floreano, J. Hallam, G. Hayes & J.-A. Meyer (Eds.), *From animals to animats 7 - The Seventh International Conf. on the Simulation of Adaptive Behavior*, The MIT Press

Matellan, V., Fernandez, C., & Molina, J. (1998). "Genetic learning of fuzzy reactive controllers." *Robotics and Autonomous Systems*, 25, 33-41

Nolfi, S., Floreano, D., Miglino, O., & Mondada, F. (1994) "How to evolve autonomous robots: Different approaches in evolutionary robotics". In R. Brooks and P. Maes, editors, *Artificial Life IV*, 190--197.

Santamaria, J. C., Sutton, R. C., & Ram, A.(1998) "Experiments with reinforcement learning in problems with continuous state and action spaces." in *Adaptive Behaviour*, 6(2).

Smart, W.D. & Kaelbling, L.P. (2002) "Effective Reinforcement Learning for Mobile Robots," *International Conf. on Robotics and Automation*, May 11-15

Stolzmann, W. (1999) "Latent Learning in Khepera Robots with Anticipatory Classifier Systems." In A.S. Wu (Ed.), *Proc. of the 1999 Genetic and Evolutionary Computation Conf. Workshop Programme*, pp. 290-297.

Stone, C. & Bull, L. (2003) "For Real! XCS with Continuous-Valued Inputs." *Evolutionary Computation* 11(3) 299-336.

Studley, M. & Bull, L. (2005) "Using the XCS Classifier System for Multi-objective Reinforcement Learning Problems". *Artificial Life* (in press).

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

Thrun, S. & Schwartz, A. (1993) "Issues in Using Function Approximation for Reinforcement Learning" In M. Mozer, P. Smolensky, D. Touretzky, J. Elman, & A. Weigend, editors, *Proc. of the Connectionist Models Summer School*, pp. 255-263, Hillsdale, NJ, 1993.

Watkins, C. (1989). *Learning from Delayed Rewards*, Thesis, University of Cambridge, England.

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

Wilson, S. (1995) "Classifier Fitness based on Accuracy." *Evolutionary Computation*, 3(2):149--175.

Wilson, S.W. (2000) "Get real! XCS with continuous-valued inputs" in Lanzi, P. L., Stolzmann, W., & Wilson, S. W., eds. *Learning Classifier Systems. From Foundations to Applications Lecture Notes in Artificial Intelligence* (LNAI-1813)

Wilson, S. W. (2001) " Mining oblique data with XCS" In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in Learning Classifier Systems. Third International Workshop (IWLCS-2000)*, Lecture Notes in Artificial Intelligence (LNAI-1996)